

# Синтаксический анализатор

## Глава 6

# Синтаксический анализатор

*Синтаксический анализатор* – часть компилятора, отвечающая за выявление и проверку синтаксических конструкций входного языка

# Функции синтаксического анализатора

- Выделение синтаксических конструкций в тексте исходной программы
- Проверка правильности каждой синтаксической конструкции
- Представление синтаксических конструкций в виде, удобном для дальнейшей генерации текста результирующей программы
- Сообщения о выявленных ошибках

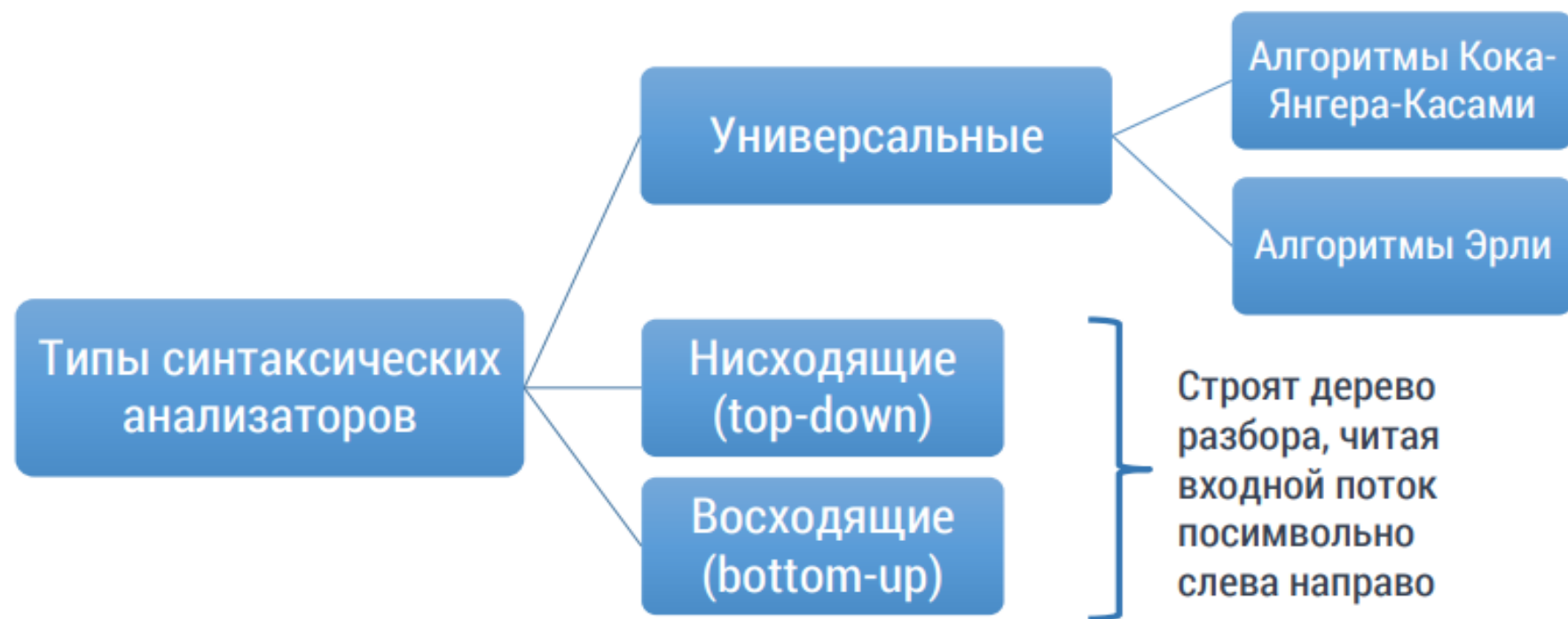
# Обработка ошибок

- *Лексические* ошибки включают неверно записанные идентификаторы, ключевые слова или операторы, или отсутствие кавычек вокруг текста, являющегося строкой
- *Синтаксические* ошибки включают неверно поставленные точки с запятой или лишние или недостающие фигурные скобки
- *Семантические* ошибки включают несоответствие типов операторов и их операндов
- *Логические* ошибки могут быть любыми — от неверных решений программиста до использования, например, в программе на языке Си оператора присваивания = вместо оператора сравнения ==

# Стратегии восстановления после ошибок

- Восстановление в режиме паники
- Восстановление на уровне фразы
- Правила ошибок
- Глобальная коррекция

# Типы синтаксических анализаторов



# Контекстно-свободные грамматики

$A \rightarrow \beta$ , где  $A \in VN$ ,  $\beta \in V^+$  (неукорачивающие)

$A \rightarrow \beta$ , где  $A \in VN$ ,  $\beta \in V^*$  (укорачивающие)

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid T / F \mid F$

$F \rightarrow (E) \mid id$

$stmt \rightarrow if (expr) stmt \text{ else } stmt$

# Почему КС-грамматики?

$$[a \dots z]([a \dots z] | [0 \dots 9])^* ((+ | - | \times | \div) [a \dots z]([a \dots z] | [0 \dots 9])^*)^*$$

a + b × c

fee ÷ fie × foe

Приоритет операторов?

$$(( \_ | \epsilon) [a \dots z]([a \dots z] | [0 \dots 9])^* ((+ | - | \times | \div) [a \dots z]([a \dots z] | [0 \dots 9])^* ( \_ | \epsilon) )^*$$

( a + b ) × c    a + ( b × c    a + b ) × c )

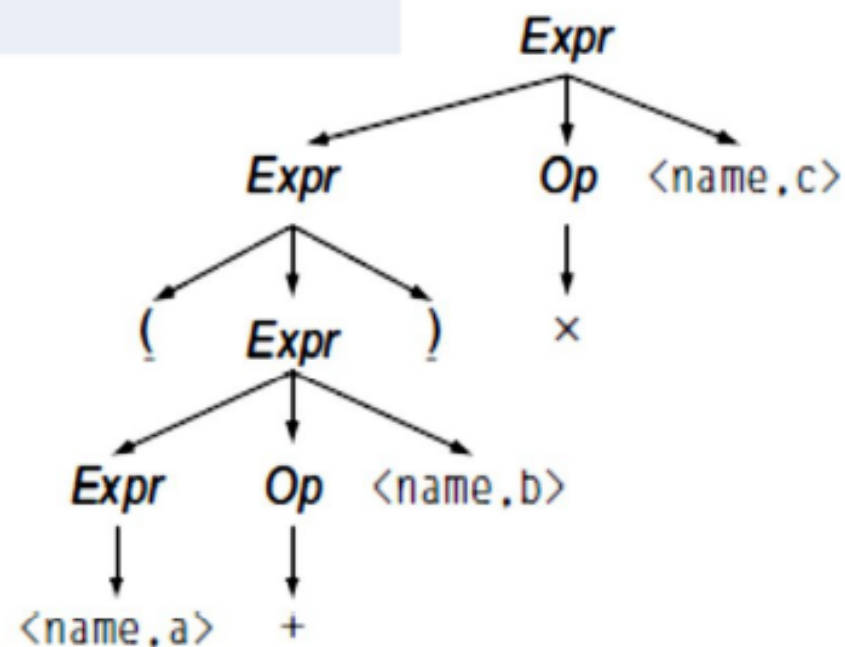


# Почему КС-грамматики?

1	$\text{Expr} \rightarrow (\text{Expr}) \mid$
2	$\text{Expr Op name} \mid$
3	$\text{name}$
4	$\text{Op} \rightarrow +$
5	$-$
6	$\times$
7	$\div$

Правило	Сентенциальная форма
	$\text{Expr}$
2	$\text{Expr Op name}$
6	$\text{Expr} \times \text{name}$
1	$(\text{Expr}) \times \text{name}$
2	$(\text{Expr Op name}) \times \text{name}$
4	$(\text{Expr} + \text{name}) \times \text{name}$
3	$(\text{name} + \text{name}) \times \text{name}$

Правосторонний вывод для  $(a + b) \times c$



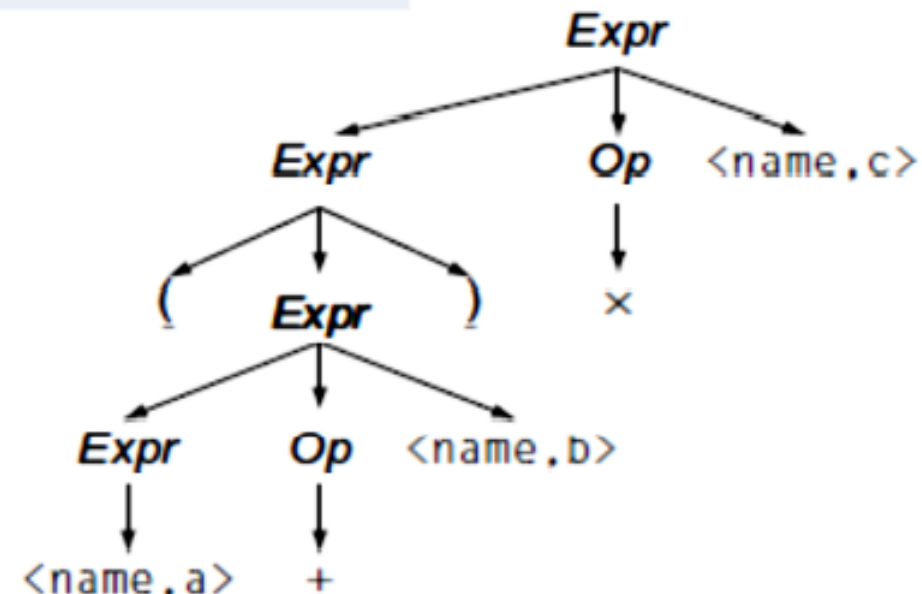
Синтаксическое дерево разбора

# Почему КС-грамматики?

1	$\text{Expr} \rightarrow (\text{Expr}) \mid$
2	$\text{Expr Op name} \mid$
3	$\text{name}$
4	$\text{Op} \rightarrow +$
5	$-$
6	$\times$
7	$\div$

Правило	Сентенциальная форма
	$\text{Expr}$
2	$\text{Expr Op name}$
1	$(\text{Expr}) \text{Op name}$
2	$(\text{Expr Op name}) \text{Op name}$
3	$(\text{name Op name}) \text{Op name}$
4	$(\text{name} + \text{name}) \text{Op name}$
6	$(\text{name} + \text{name}) \times \text{name}$

Левосторонний вывод для  $(a + b) \times c$

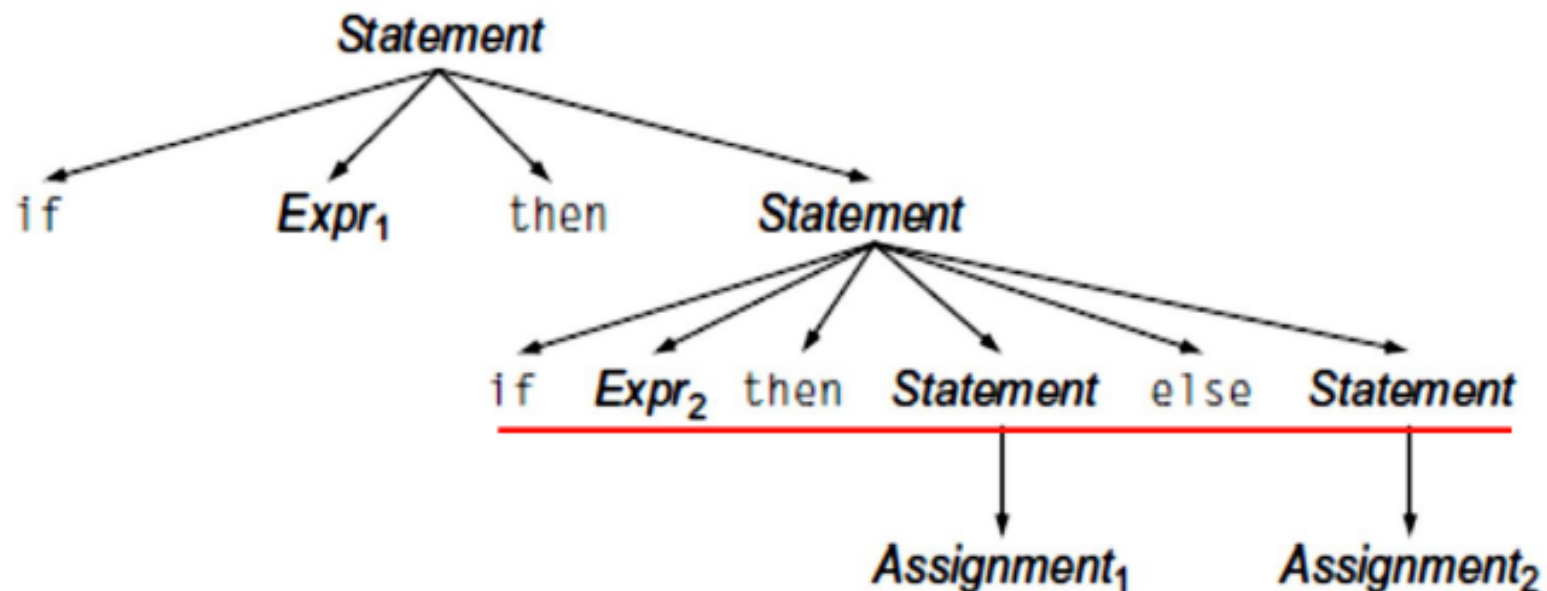


Синтаксическое дерево разбора

# Неоднозначные КС-грамматики

1	Statement $\rightarrow$ if Expr then Statement else Statement
2	if Expr then Statement
3	Assignment
4	... other statements ...

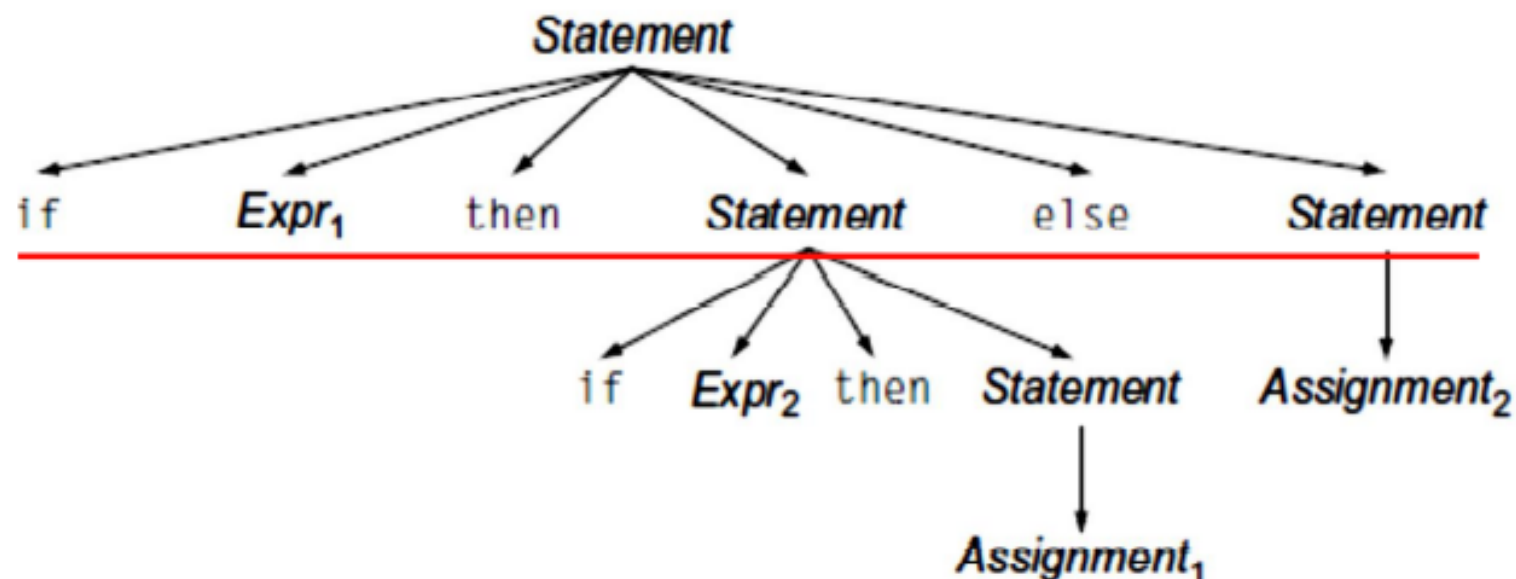
if *Expr*<sub>1</sub> then if *Expr*<sub>2</sub> then *Assignment*<sub>1</sub> else *Assignment*<sub>2</sub>



# Неоднозначные КС-грамматики

1	Statement $\rightarrow$ if Expr then Statement else Statement
2	if Expr then Statement
3	Assignment
4	... other statements ...

if Expr<sub>1</sub> then if Expr<sub>2</sub> then Assignment<sub>1</sub> else Assignment<sub>2</sub>



# Неоднозначные КС-грамматики

1	Statement $\rightarrow$ if Expr then Statement
2	if Expr then WithElse else Statement
3	Assignment
4	WithElse $\rightarrow$ if Expr then WithElse else WithElse
5	Assignment

Правило	Сентенциальная форма
	Statement
1	if Expr then Statement
2	if Expr then if Expr then WithElse else Statement
3	if Expr then if Expr then WithElse else Assignment
5	if Expr then if Expr then Assignment else Assignment

# Задача синтаксического анализа

Построение синтаксического дерева  
разбора

Нисходящий  
синтаксический  
анализ

Восходящий  
синтаксический  
анализ

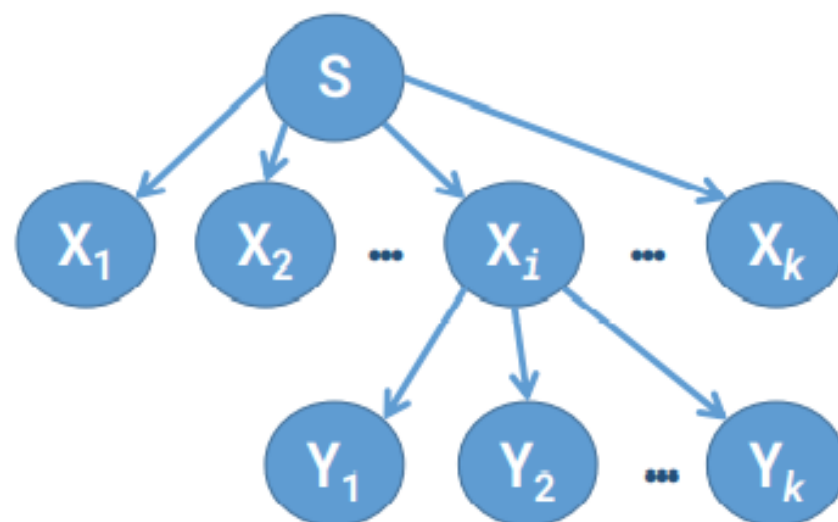
# Нисходящий синтаксический анализ

При нисходящем анализе входная цепочка символов просматривается слева направо и порождается левосторонний вывод



Метод рекурсивного спуска

$$\begin{aligned} S &\rightarrow X_1 X_2 \dots X_{i-1} X_i X_{i+1} \dots X_k \\ X_i &\rightarrow Y_1 Y_2 \dots Y_l \end{aligned}$$



# Псевдокод разбора для нетерминала

```
void S()
{
    Выбираем S-правило  $S \rightarrow X_1X_2...X_k$ ;
    for (i от 1 до k)
    {
        if ( $X_i$  – нетерминал)
            Вызов процедуры  $X_i()$ ;
        else if ( $X_i$  равно текущему входному символу)
            Переходим к следующему входному символу;
        else /* Обнаружена ошибка */;
    }
}
```



# Условия применимости метода рекурсивного спуска

Два варианта правил в КС-грамматике:

1.  $A \rightarrow \gamma, \gamma \in (\mathbf{VT} \cup \mathbf{VN})^*$  и это единственное правило для  $A$

2.  $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n, a_i \in \mathbf{VT}, \beta_i \in (\mathbf{VT} \cup \mathbf{VN})^*, a_i \neq a_j$

# Рекурсия

Символ  $A \in VN$  называется рекурсивным, если для него существует цепочка вывода вида

$A \Rightarrow^+ \alpha A \beta$ , где  $\alpha, \beta \in (VT \cup VN)^*$

Если  $\alpha = \varepsilon$  и  $\beta \neq \varepsilon$ , то рекурсия называется *левой*.

Если  $\alpha \neq \varepsilon$  и  $\beta = \varepsilon$ , то рекурсия называется *правой*.

Если  $\alpha = \varepsilon$  и  $\beta = \varepsilon$ , то рекурсия представляет собой *цикл*.

# Левая рекурсия

Правило	Сентенциальная форма	Входная цепочка символов
	Expr	$\uparrow \text{name} + \text{name} \times \text{name}$
1	Expr + Term	$\uparrow \text{name} + \text{name} \times \text{name}$
1	Expr + Term + Term	$\uparrow \text{name} + \text{name} \times \text{name}$
1	...	$\uparrow \text{name} + \text{name} \times \text{name}$

0	Goal $\rightarrow$ Expr
1	Expr $\rightarrow$ Expr + Term
2	Expr $\rightarrow$ Expr - Term
3	Term
4	Term $\rightarrow$ Term $\times$ Factor
5	Term $\rightarrow$ Term $\div$ Factor
6	Factor
7	Factor $\rightarrow$ (Expr)
8	num
9	name

# Алгоритм устранения левой рекурсии

1.  $VN = \{A_1, A_2, \dots, A_n\}, i = 1$

2. Правила для  $A_i$ .

Если в правилах нет левой рекурсии, то они записываются в  $P'$  без изменений, а  $A_i$  добавляется в  $VN'$ .

Иначе правила для  $A_i$  записываются в виде

$A_i \rightarrow A_i \alpha_1 \mid A_i \alpha_2 \mid \dots \mid A_i \alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_p$ , где ни одна из цепочек  $\beta_j$  не начинается с  $A_k$  таких, что  $k \leq i$ .

Вместо этих правил в  $P'$  записываются правила вида:

$$A_i \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_p \mid \beta_1 A'_i \mid \beta_2 A'_i \mid \dots \mid \beta_p A'_i$$

$$A'_i \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m \mid \alpha_1 A'_i \mid \alpha_2 A'_i \mid \dots \mid \alpha_m A'_i$$

$A_i$  и  $A'_i$  включаются в  $VN$ .

3. Если  $i = n$ , то  $G'$  построена, перейти к 6, иначе  $i = i + 1, j = 1$ , перейти к 4

## Алгоритм устранения левой рекурсии

4. Для  $A_j$  в  $P'$  заменить все правила вида  $A_j \rightarrow A_j \alpha$ , где  $\alpha \in (VT \cup VN)^*$ , на правила вида  $A_j \rightarrow \beta_1 \alpha \mid \beta_2 \alpha \mid \dots \mid \beta_m \alpha$ , причем  $A_j \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$  – все правила для  $A_j$
5. Если  $j = i - 1$ , перейти к 2, иначе  $j = j + 1$  и перейти к 4
6. Целевым символом грамматики  $G'$  становится  $A_k$ , соответствующий символу  $S$  исходной грамматики

# Пример

$G(\{+, -, /, *, a, b\}, \{S, T, E\}, P, S)$

$P: S \rightarrow S + T \mid S - T \mid T$

$T \rightarrow T * E \mid T / E \mid E$

$E \rightarrow (S) \mid a \mid b$

$VN' = \{A_1, A'_1, A_2, A'_2, A_3\}$

1.  $VN' = \{A_1, A_2, A_3\}, i = 1$

$A_1 \rightarrow A_1 + A_2 \mid A_1 - A_2 \mid A_2$

$A_2 \rightarrow A_2 * A_3 \mid A_2 / A_3 \mid A_3$

$A_3 \rightarrow (A_1) \mid a \mid b$

2.  $A_1 \rightarrow A_1 + A_2 \mid A_1 - A_2 \mid A_2$

$A_1 \rightarrow A_1 \alpha_1 \mid A_1 \alpha_2 \mid \beta_1$

$\alpha_1 = +A_2, \alpha_2 = -A_2, \beta_1 = A_2$

$A_1 \rightarrow A_2 \mid A_2 A'_1$

$A'_1 \rightarrow +A_2 \mid -A_2 \mid +A_2 A'_1 \mid -A_2 A'_1$

3.  $A_2 \rightarrow A_2 * A_3 \mid A_2 / A_3 \mid A_3$

$A_2 \rightarrow A_2 \alpha_1 \mid A_2 \alpha_2 \mid \beta_1$

$\alpha_1 = *A_3, \alpha_2 = /A_3, \beta_1 = A_3$

$A_2 \rightarrow A_3 \mid A_3 A'_2$

$A'_2 \rightarrow *A_3 \mid /A_3 \mid *A_3 A'_2 \mid /A_3 A'_2$

# Пример (продолжение)

4.  $A_3 \rightarrow (A_1) \mid a \mid b$  – нет левой рекурсии

$G(\{+, -, /, *, a, b\}, \{A_1, A'_1, A_2, A'_2, A_3\}, P, A_1)$

$P: A_1 \rightarrow A_2 \mid A_2 A'_1$

$A'_1 \rightarrow + A_2 \mid - A_2 \mid + A_2 A'_1 \mid - A_2 A'_1$

$A_2 \rightarrow A_3 \mid A_3 A'_2$

$A'_2 \rightarrow * A_3 \mid / A_3 \mid * A_3 A'_2 \mid / A_3 A'_2$

$A_3 \rightarrow (A_1) \mid a \mid b$

# Устранение левой рекурсии

До устранения	После устранения
$\text{Expr} \rightarrow \text{Expr} + \text{Term} \mid$ $\text{Expr} - \text{Term} \mid$ $\text{Term}$	$\text{Expr} \rightarrow \text{Term Expr}'$ $\text{Expr}' \rightarrow + \text{Term Expr}' \mid$ $- \text{Term Expr}' \mid$ $\epsilon$
$\text{Term} \rightarrow \text{Term} \times \text{Factor} \mid$ $\text{Term} \div \text{Factor} \mid$ $\text{Factor}$	$\text{Term} \rightarrow \text{Factor Term}'$ $\text{Term}' \rightarrow \times \text{Factor Term}' \mid$ $\div \text{Factor Term}' \mid$ $\epsilon$

0	$\text{Goal} \rightarrow \text{Expr}$	6	$\text{Term}' \rightarrow \times \text{Factor Term}' \mid$
1	$\text{Expr} \rightarrow \text{Term Expr}'$	7	$\div \text{Factor Term}' \mid$
2	$\text{Expr}' \rightarrow + \text{Term Expr}' \mid$	8	$\epsilon$
3	$- \text{Term Expr}' \mid$	9	$\text{Factor} \rightarrow (\text{Expr}) \mid$
4	$\epsilon$	10	$\text{num} \mid$
5	$\text{Term} \rightarrow \text{Factor Term}'$	11	$\text{name}$



Правило	Сентенциальная форма	Входная цепочка символов
	Expr	name + name × name
1	Term Expr'	name + name × name
5	Factor Term' Expr'	name + name × name
11	name Term' Expr'	name + name × name
→	name Term' Expr'	name + name × name
8	name Expr'	name + name × name
2	name + Term Expr'	name + name × name
→	name + Term Expr'	name + name × name
5	name + Factor Term' Expr'	name + name × name
11	name + name Term' Expr'	name + name × name
→	name + name Term' Expr'	name + name × name
6	name + name × Factor Term' Expr'	name + name × name
→	name + name × Factor Term' Expr'	name + name × name
11	name + name × name Term' Expr'	name + name × name
→	name + name × name Term' Expr'	name + name × name
8	name + name × name Expr'	name + name × name
4	name + name × name	name + name × name

# Левая факторизация

Это преобразование позволяет исключить из КС-грамматики правила, имеющие в своих правых частях одинаковые префиксы

$\text{Stmt} \rightarrow \underline{\text{if Expr then Stmt}} \text{ else Stmt} \mid \underline{\text{if Expr then Stmt}}$

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$



$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 \mid \beta_2$

# Алгоритм левой факторизации

**Вход:** грамматика  $G$

**Выход:** эквивалентная левофакторизованная грамматика

**Метод:** для каждого нетерминала  $A$  находим самый длинный префикс  $\alpha$ , общий для двух или большего числа альтернатив. Заменяем все правила  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$ , где  $\gamma$  представляет все альтернативы, не начинающиеся с  $\alpha$ , правилами

$$\begin{aligned} A &\rightarrow \alpha A' \mid \gamma \\ A' &\rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \end{aligned}$$

Выполнять преобразование до тех пор, пока в грамматике остаются правила с одинаковыми префиксами

# Пример

Stmt  $\rightarrow$  if Expr then Stmt | if Expr then Stmt else Stmt | a  
Expr  $\rightarrow$  b



Stmt  $\rightarrow$  if Expr then Stmt Stmt' | a  
Stmt'  $\rightarrow$  else Stmt |  $\epsilon$   
Expr  $\rightarrow$  b

## Пример реализации метода рекурсивного спуска

$S \rightarrow aA \mid bB$

$A \rightarrow a \mid bA \mid cC$

$B \rightarrow b \mid aB \mid cC$

$C \rightarrow AaBb$

Грамматика удовлетворяет  
условию применимости  
метода рекурсивного спуска

```
int main()
{
    ...
    if(S()) return 0;
    else return -1;
}
int S()
{
    c = gc();
    if(c == 'a'){
        if(!A()) return 0;
    }else if(c == 'b'){
        if(!B()) return 0;
    }else{
        return 0;
    }
    return 1;
}
```

## Пример реализации метода рекурсивного спуска

$S \rightarrow aA \mid bB$

$A \rightarrow a \mid bA \mid cC$

$B \rightarrow b \mid aB \mid cC$

$C \rightarrow AaBb$

Грамматика удовлетворяет  
условию применимости  
метода рекурсивного спуска

```
int A()  
{  
    c = gc();  
    if(c == 'b')  
    {  
        if(!A()) return 0;  
    }else if(c == 'c')  
    {  
        if(!C()) return 0;  
    }else if(c == 'a')  
    {  
        return 1;  
    }else{  
        return 0;  
    }  
    return 1;  
}
```

## Пример реализации метода рекурсивного спуска

$S \rightarrow aA \mid bB$

$A \rightarrow a \mid bA \mid cC$

$B \rightarrow b \mid aB \mid cC$

$C \rightarrow AaBb$

Грамматика удовлетворяет  
условию применимости  
метода рекурсивного спуска

```
int B()  
{  
    c = gc();  
    if(c == 'a')  
    {  
        if(!B()) return 0;  
    }else if(c == 'c')  
    {  
        if(!C()) return 0;  
    }else if(c == 'b')  
    {  
        return 1;  
    }else{  
        return 0;  
    }  
    return 1;  
}
```

# Пример реализации метода рекурсивного спуска

$S \rightarrow aA \mid bB$

$A \rightarrow a \mid bA \mid cC$

$B \rightarrow b \mid aB \mid cC$

$C \rightarrow AaBb$

Грамматика удовлетворяет  
условию применимости  
метода рекурсивного спуска

```
int C()  
{  
    if(!A()) return 0;  
    c = gc();  
    if(c == 'a')  
    {  
        if(!B()) return 0;  
        c = gc();  
        if(c != 'b')  
        {  
            return 0;  
        }  
    }else{  
        return 0;  
    }  
    return 1;  
}
```



## Пример реализации метода рекурсивного спуска

$S \rightarrow S + T \mid S - T \mid T$

$T \rightarrow T * E \mid T / E \mid E$

$E \rightarrow (S) \mid a \mid b$



$S \rightarrow T \{ + T \mid - T \}$

$T \rightarrow E \{ * E \mid / E \}$

$E \rightarrow (S) \mid a \mid b$

```
int S()
{
    if(!T()) return 0;
    while(gc() == '+' ||
          gc() == '-')
    {
        if(!T()) return 0;
    }
    return 1;
}
```