Формальные языки и грамматики

Цепочки символов

Цепочка символов – произвольная упорядоченная конечная последовательность символов, записанных один за другим

Обозначение: α, β, γ…

Символ – базовое понятие в теории формальных языков

Цепочки символов

Две цепочки символов *равны*, если они имеют один и тот же состав символов, одно и то же их количество и одинаковый порядок следования символов в цепочке

Длина цепочки символов = количеству символов в ней ($|\alpha|$)

Если $\alpha = \beta$, то $|\alpha| = |\beta|$

Операции над цепочками символов

- *Конкатенация* – дописывание второй цепочки в конец первой

$\alpha = ab,\ \beta = cd$ → $\alpha\beta = abcd$

$\alpha\beta \neq \beta\alpha;\ (\alpha\beta)\gamma = \alpha(\beta\gamma)$

- *Замена* (подстановка) – замена одной из подцепочек цепочки символов на другую цепочку символов

$\gamma = \alpha\nu\beta$ → $\alpha = a,\ \nu = aba,\ \beta = cd$ → $\gamma = aabacd$

Операции над цепочками символов

- *Обращение* – запись символов цепочки в обратном порядке

$(\alpha\beta)^R = \beta^R\alpha^R$

α = abcd → α^R = dcba

- *Итерация* – повторение цепочки *n* раз, где $n \in N$, $n > 0$

α^n $\quad \alpha^1 = \alpha \quad \alpha^2 = \alpha\alpha \quad \alpha^3 = \alpha\alpha\alpha \quad \ldots$

Пустая цепочка символов

Пустая цепочка символов – цепочка, не содержащая ни одного символа (ε)

$$|\varepsilon| = 0$$

$$\varepsilon\alpha = \alpha\varepsilon = \alpha$$

$$\varepsilon^R = \varepsilon$$

$$\varepsilon^n = \varepsilon;\ n \geq 0$$

$$\alpha^0 = \varepsilon$$

Понятие языка

- *Алфавит* – счётное множество допустимых символов языка (**V**)

- Цепочка символов α является *цепочкой над алфавитом* **V** (α(**V**)), если в неё входят только символы, принадлежащие **V**

$\mathbf{V}^+$ – множество всех цепочек над алфавитом **V** без ε

$\mathbf{V}^*$ – множество всех цепочек над алфавитом **V**, включая ε : $\mathbf{V}^* = \mathbf{V}^+ \cup \{\varepsilon\}$

Понятие языка

- *Язык* L над алфавитом V (L(V)) – некоторое счётное подмножество цепочек конечной длины из множества всех цепочек над алфавитом V

  $L'(V) \subseteq L(V)$, если $\forall \alpha \in L(V)$: $\alpha \in L'(V)$
  $L'(V) = L(V)$, если $L'(V) \subseteq L(V)$ и $L(V) \subseteq L'(V)$
  $L'(V) \cong L(V)$, если $L'(V) \cup \{\varepsilon\} = L(V)$

- *Предложение* – цепочка символов, принадлежащая заданному языку

Грамматика

- *Грамматика* – описание способа построения предложений некоторого языка
- *Правило* – упорядоченная пара цепочек символов

Формальное определение грамматики

G (VT, VN, P, S) :

- VT – множество терминальных символов (символы алфавита языка)
- VN – множество нетерминальных символов (слова, понятия, конструкции языка)

  $VN \cap VT = \varnothing$
- P – множество правил вида $\alpha \to \beta$, где $\alpha \in (VN \cup VT)^+$, $\beta \in (VN \cup VT)^*$
- S – целевой (начальный) символ грамматики, $S \in VN$

Эквивалентность грамматик

$L(G)$ – язык, заданный грамматикой G

$L(G')$ – язык, заданный грамматикой G'

$G = G'$, если $L(G) = L(G')$

$G \cong G'$, если $L(G) \cong L(G')$

Форма Бэкуса-Наура (БНФ) Backus-Naur Form (BNF)

$$\alpha \to \beta_1,\ \alpha \to \beta_2,\ \ldots,\ \alpha \to \beta_n$$

$$\langle\alpha\rangle \to \langle\beta_1\rangle | \langle\beta_2\rangle | \ldots | \langle\beta_n\rangle$$

G ({0,1,2,3,4,5,6,7,8,9,+,–}, {<целое число>,<целое без знака>,<цифра>}, P, <целое число>)

P:

<целое число> → <целое без знака> | +<целое без знака> | –<целое без знака>

<целое без знака> → <цифра> | <целое без знака><цифра>

<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Расширенная форма Бэкуса-Наура (РБНФ)

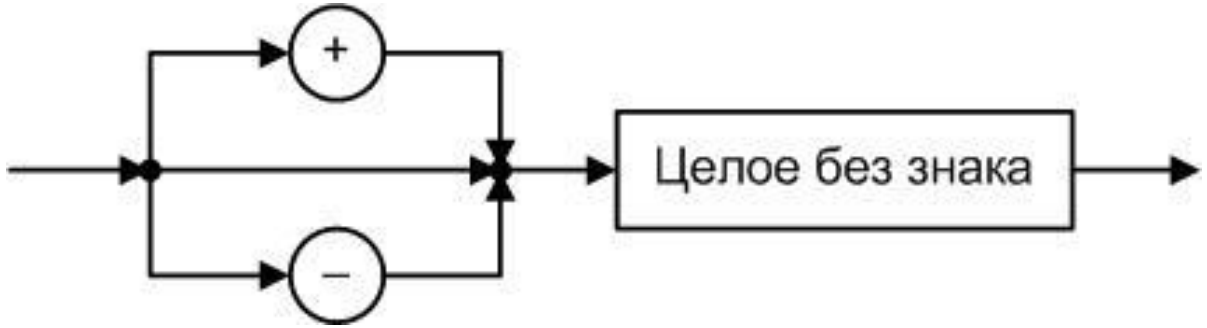
- необязательные элементы заключаются в [ ]
- повторяющиеся элементы заключаются в { }

```
<целое число> → [ + | – ] <целое без знака>

<целое без знака> → <цифра> { <цифра> }

<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

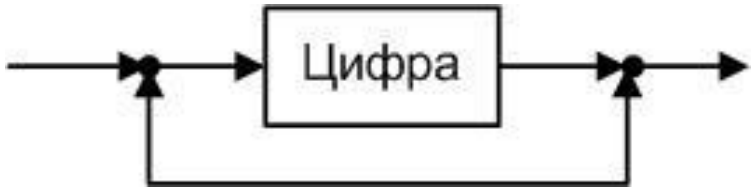
Синтаксические диаграммы Вирта

Предложены Н. Виртом для описания синтаксиса языка Pascal

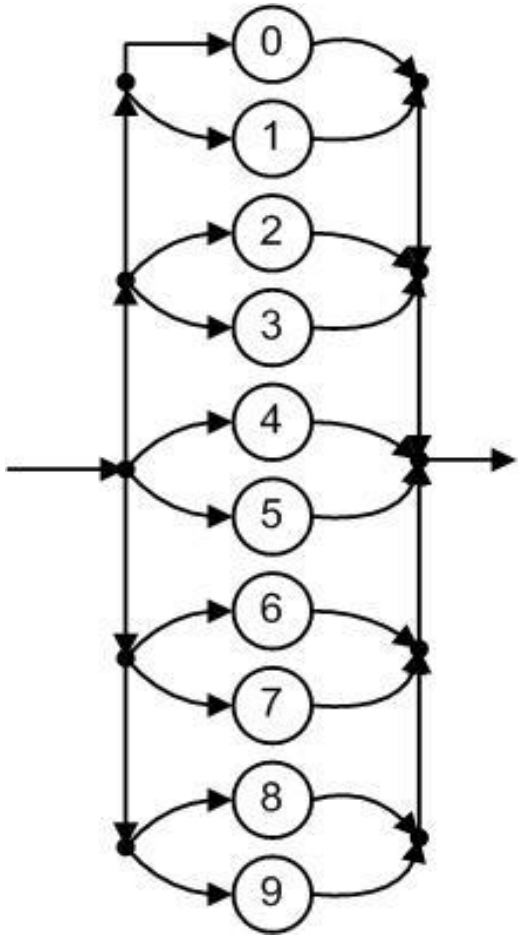
Целое число:



Целое без знака:



Цифра:



Классификация грамматик по Н. Хомскому

- **Тип 0:** *Грамматики с фразовой структурой*

  $\alpha \to \beta$, **где** $\alpha \in \mathbf{V}^+$, $\beta \in \mathbf{V}^*$

  <u>Пример:</u>

  G({0, 1}, {A, S}, **P**, S)

  P:

  $S \to 0A1$

  $0A \to 00A1$

  $A \to \varepsilon$

Классификация грамматик по Н. Хомскому

- **Тип 1:** *Контекстно-зависимые (КЗ-грамматики)*

  $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, **где** $\alpha_1, \alpha_2 \in \mathbf{V}^*$, $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

  Неукорачивающие грамматики

  $\alpha \to \beta$, **где** $\alpha, \beta \in \mathbf{V}^+$, $|\beta| \geq |\alpha|$

<u>Пример</u>: **G**({a, b, c}, {B, C, S}, **P**, S)

P: $S \to aSBC$
$S \to abC$
$CB \to BC$
$bB \to bb$
$bC \to bc$
$cB \to bc$
$cC \to cc$

Классификация грамматик по Н. Хомскому

- **Тип 2**: *Контекстно-свободные (КС-грамматики)*

  $A \to \beta$, где $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$ (неукорачивающие)

  $A \to \beta$, где $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^*$ (укорачивающие)

  <u>Пример</u>:

  $\mathbf{G}(\{i, +, *, (,)\}, \{E, T, P\}, \mathbf{P}, E)$

  $\mathbf{P}$: $E \to E + T \mid T$

  $T \to T * P \mid P$

  $P \to i \mid (E)$

Классификация грамматик по Н. Хомскому

- **Тип 3**: *Регулярные (Р-грамматики)*

$A \to B\gamma$ или $A \to \gamma$, где $A, B \in \mathbf{VN}$, $\gamma \in \mathbf{VT}^*$ (леволинейные)

$A \to \gamma B$ или $A \to \gamma$, где $A, B \in \mathbf{VN}$, $\gamma \in \mathbf{VT}^*$ (праволинейные)

<u>Пример</u>:

$\mathbf{G}(\{0, 1\}, \{A, S\}, \mathbf{P}, S)$

$\mathbf{P}$: $S \to 0A$

$A \to 1A \mid 0A \mid 0 \mid 1$

Пример

G ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, –}, {<целое число>, <целое без знака> , <цифра>}, **P**, <целое число>)

P:

<целое число> → <целое без знака> | + <целое без знака> | – <целое без знака>

<целое без знака> → <цифра> | <целое без знака><цифра>

<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Тип грамматики?

Пример

G ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, –}, {<целое число>, <целое без знака> , <цифра>}, **P**, <целое число>)

P:

<целое число> → <целое без знака> | + <целое без знака> | – <целое без знака>

<целое без знака> → <цифра> | <целое без знака><цифра>

<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Тип грамматики?

Тип 0: $\alpha \to \beta$, где $\alpha \in \mathbf{V}^+$, $\beta \in \mathbf{V}^*$

Пример

G ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, –}, {<целое число>, <целое без знака> , <цифра>}, **P**, <целое число>)

P:

<целое число> $\rightarrow$ <целое без знака> | + <целое без знака> | – <целое без знака>

<целое без знака> $\rightarrow$ <цифра> | <целое без знака><цифра>

<цифра> $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Тип грамматики?

Тип 0: $\alpha \rightarrow \beta$, где $\alpha \in \mathbf{V}^+$, $\beta \in \mathbf{V}^*$

Тип 1: $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, где $\alpha_1, \alpha_2 \in \mathbf{V}^*$, $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

$\alpha \rightarrow \beta$, где $\alpha, \beta \in \mathbf{V}^+$, $|\beta| \geq |\alpha|$

Пример

G ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, –}, {<целое число>, <целое без знака> , <цифра>}, **P**, <целое число>)

P:

<целое число> $\rightarrow$ <целое без знака> | + <целое без знака> | – <целое без знака>

<целое без знака> $\rightarrow$ <цифра> | <целое без знака><цифра>

<цифра> $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Тип грамматики?

Тип 0: $\alpha \rightarrow \beta$, где $\alpha \in \mathbf{V}^+$, $\beta \in \mathbf{V}^*$

Тип 1: $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, где $\alpha_1, \alpha_2 \in \mathbf{V}^*$, $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

$\alpha \rightarrow \beta$, где $\alpha, \beta \in \mathbf{V}^+$, $|\beta| \geq |\alpha|$

Тип 2: $A \rightarrow \beta$, где $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

Пример

G ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, –}, {<целое число>, <целое без знака> , <цифра>}, **P**, <целое число>)

P:

<целое число> $\rightarrow$ <целое без знака> | + <целое без знака> | – <целое без знака>

<целое без знака> $\rightarrow$ <цифра> | <целое без знака><цифра>

<цифра> $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Тип грамматики?

✓ Тип 0: $\alpha \rightarrow \beta$, где $\alpha \in \mathbf{V}^+$, $\beta \in \mathbf{V}^*$

✓ Тип 1: $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, где $\alpha_1, \alpha_2 \in \mathbf{V}^*$, $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

$\alpha \rightarrow \beta$, где $\alpha, \beta \in \mathbf{V}^+$, $|\beta| \geq |\alpha|$

✓ Тип 2: $A \rightarrow \beta$, где $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

— Тип 3: $A \rightarrow B\gamma$ или $A \rightarrow \gamma$, где $A, B \in \mathbf{VN}$, $\gamma \in \mathbf{VT}^*$

$A \rightarrow \gamma B$ или $A \rightarrow \gamma$, где $A, B \in \mathbf{VN}$, $\gamma \in \mathbf{VT}^*$

Пример

G ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, –}, {<целое число>, <целое без знака> , <цифра>}, **P**, <целое число>)

P:

<целое число> → <целое без знака> | + <целое без знака> | – <целое без знака>

<целое без знака> → <цифра> | <целое без знака><цифра>

<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Тип грамматики?

✓ Тип 0: $\alpha \to \beta$, где $\alpha \in \mathbf{V}^+$, $\beta \in \mathbf{V}^*$

✓ Тип 1: $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, где $\alpha_1, \alpha_2 \in \mathbf{V}^*$, $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

$\alpha \to \beta$, где $\alpha, \beta \in \mathbf{V}^+$, $|\beta| \geq |\alpha|$

✓ Тип 2: $A \to \beta$, где $A \in \mathbf{VN}$, $\beta \in \mathbf{V}^+$

~~Тип 3: $A \to B\gamma$ или $A \to \gamma$, где $A, B \in \mathbf{VN}$, $\gamma \in \mathbf{VT}^*$~~

~~$A \to \gamma B$ или $A \to \gamma$, где $A, B \in \mathbf{VN}$, $\gamma \in \mathbf{VT}^*$~~

Классификация языков

Языки классифицируются в соответствии с типами грамматик, с помощью которых они заданы

- **Тип 0** : языки с фразовой структурой
- **Тип 1** : контекстно-зависимые (КЗ) языки
- **Тип 2** : контекстно-свободные (КС) языки
- **Тип 3** : регулярные языки

Классификация языков

Один и тот же язык может быть задан сколь угодно большим количеством грамматик, которые могут относиться к различным типам

Пример:

Язык L задан грамматиками G_1, G_2, относящимися к типу 1 (КЗ), грамматикой G_3 (тип 2 (КС)) и грамматикой G_4 (тип 3 (Р)).

Тип языка – регулярный

Пример

- Задан язык L = $\{0^n1^m \mid n, m > 0\}$.
  Написать правила грамматик разных типов, задающих этот язык.

Тип 3 (Р)	Тип 2 (КС)	Тип 1 (КЗ)	Тип 0
$S \to 0A$ $A \to 0A \mid 1B \mid 1$ $B \to 1B \mid 1$	$S \to AB$ $A \to 0A \mid 0$ $B \to 1B \mid 1$	$S \to AB$ $A \to 0A \mid 0$ $0A \to 00A$ $B \to 1B \mid 1$	$S \to 0AB1$ $A \to 0A \mid \varepsilon$ $0A \to 00A$ $B \to 1B \mid \varepsilon$
$S \to B1$ $B \to B1 \mid A0 \mid 0$ $A \to A0 \mid 0$	$S \to AB$ $A \to A0 \mid 0$ $B \to B1 \mid 1$	$S \to AB$ $A \to 0A \mid 0$ $0A \to 00A$ $B \to 1B \mid 1$ $1B \to 11B$	$S \to 0AB1$ $A \to 0A \mid \varepsilon$ $0A \to 00A$ $B \to 1B \mid \varepsilon$ $1B \to 11B$

Пример

G ({a, b, c}, {S}, **P**, S)

P:

$S \to aSa \mid bSb \mid c$

Какой язык порождает данная грамматика?

$S \Rightarrow c$

$S \Rightarrow bSb \Rightarrow bcb$

$S \Rightarrow aSa \Rightarrow aca$

$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabcbaa$

$S \Rightarrow bSb \Rightarrow baSab \Rightarrow baaSaab \Rightarrow baacaab$

L(G) = $\{\alpha c \alpha^R \mid \alpha \in \{a, b\}^*\}$, где α^R – обращение цепочки α

Цепочки вывода

- *Вывод* – процесс порождения предложения языка на основе правил определяющей язык грамматики
- Цепочка $\beta = \delta_1\gamma\delta_2$ называется *непосредственно выводимой* из цепочки $\alpha = \delta_1\omega\delta_2$, если в грамматике существует правило

  $\omega \rightarrow \gamma$

  $\alpha \Rightarrow \beta$
- Цепочка β называется *выводимой* из цепочки α ($\alpha \Rightarrow^* \beta$), если выполняется одно из условий:
  - $\alpha \Rightarrow \beta$
  - $\alpha \Rightarrow^* \gamma$ и $\gamma \Rightarrow \beta$

  $\alpha \Rightarrow \gamma_1 \Rightarrow \ldots \Rightarrow \gamma_i \Rightarrow \ldots \Rightarrow \gamma_n \Rightarrow \beta$ – *цепочка вывода*
- Если цепочка вывода из α в β содержит одну или более промежуточных цепочек, то β *нетривиально выводима* из α

  $\alpha \Rightarrow^+ \beta$

Пример

G({i, +, *, (,)}, {E, T, P}, **P**, E)

P: $E \to E + T \mid T$

$T \to T * P \mid P$

$P \to i \mid (E)$

1. $E \Rightarrow E + T \Rightarrow T + T \Rightarrow P + T \Rightarrow i + T \Rightarrow i + T * P \Rightarrow i + P * P \Rightarrow i + i * P \Rightarrow \mathbf{i + i * i}$
2. $E \Rightarrow E + T \Rightarrow E + T * P \Rightarrow E + T * i \Rightarrow E + P * i \Rightarrow E + i * i \Rightarrow T + i * i \Rightarrow P + i * i \Rightarrow \mathbf{i + i * i}$
3. $E \Rightarrow E + T \Rightarrow E + T * P \Rightarrow T + T * P \Rightarrow T + T * i \Rightarrow P + T * i \Rightarrow P + P * i \Rightarrow i + P * i \Rightarrow \mathbf{i + i * i}$

Сентенциальная форма

- Вывод называется *законченным*, если на основе цепочки, полученной в результате этого вывода, нельзя больше сделать ни одного шага вывода

- Цепочка, полученная в результате законченного вывода, называется *конечной* цепочкой вывода

- Цепочка символов $\alpha \in$ **V*** называется *сентенциальной формой грамматики* **G**(**VT**, **VN**, **P**, S), **V = VT**$\cup$**VN**, если она выводима из целевого символа грамматики S: S $\Rightarrow^*$ β

- Если цепочка $\alpha \in$ **VT*** получена в результате законченного вывода, то она называется *конечной сентенциальной формой*

Пример

G({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, –}, {S, T, F}, **P**, S)

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

$T \to F \mid TF$

$F \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

1. $S \Rightarrow -T \Rightarrow -TF \Rightarrow -TFF \Rightarrow -FFF \Rightarrow -4FF \Rightarrow -47F \Rightarrow -479$
2. $S \Rightarrow T \Rightarrow TF \Rightarrow T8 \Rightarrow F8 \Rightarrow 18$
3. $T \Rightarrow TF \Rightarrow T0 \Rightarrow TF0 \Rightarrow T50 \Rightarrow F50 \Rightarrow 350$
4. $TFT \Rightarrow TFFT \Rightarrow TFFF \Rightarrow FFFF \Rightarrow 1FFF \Rightarrow 1FF4 \Rightarrow 10F4 \Rightarrow 1004$
5. $F \Rightarrow 5$

Левосторонний и правосторонний выводы

- Вывод называется *левосторонним*, если в нем на каждом шаге вывода правило грамматики всегда применяется к *крайнему левому нетерминальному символу* в цепочке

- Вывод называется *правосторонним*, если в нем на каждом шаге вывода правило грамматики всегда применяется к *крайнему правому нетерминальному символу* в цепочке

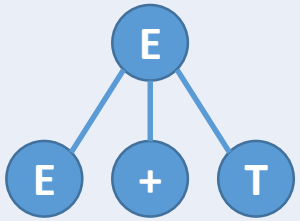
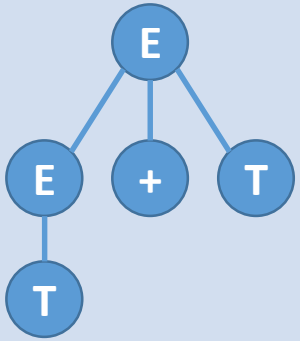
Дерево вывода

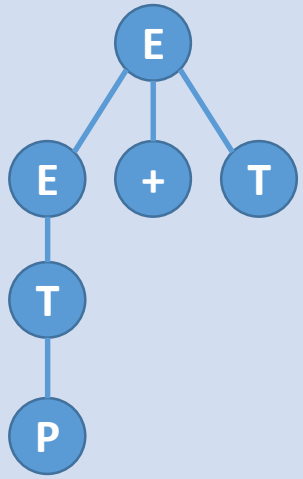
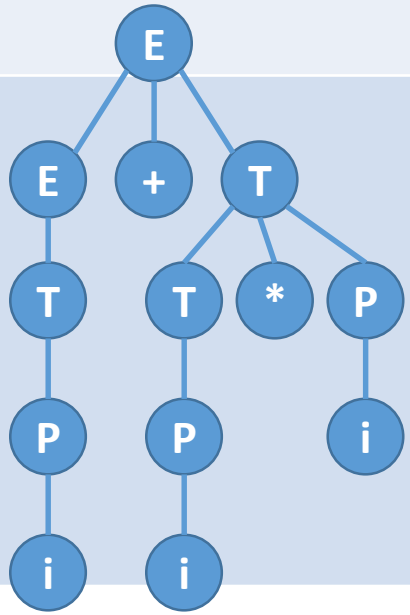
Деревом вывода грамматики **G**(**VT**, **VN**, **P**, S) называется дерево, которое соответствует некоторой цепочке вывода и удовлетворяет следующим условиям:

- ✓ каждая вершина дерева обозначается символом грамматики $A \in (\mathbf{VT} \cup \mathbf{VN} \cup \{\varepsilon\})$;
- ✓ корнем дерева является вершина, обозначенная целевым символом грамматики;
- ✓ листьями дерева являются вершины, обозначенные терминальными символами грамматики или символом ε;
- ✓ каждая ветвь дерева соответствует правилу грамматики.

Пример

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow P + T \Rightarrow i + T \Rightarrow i + T * P \Rightarrow i + P * P \Rightarrow i + i * P \Rightarrow i + i * i$

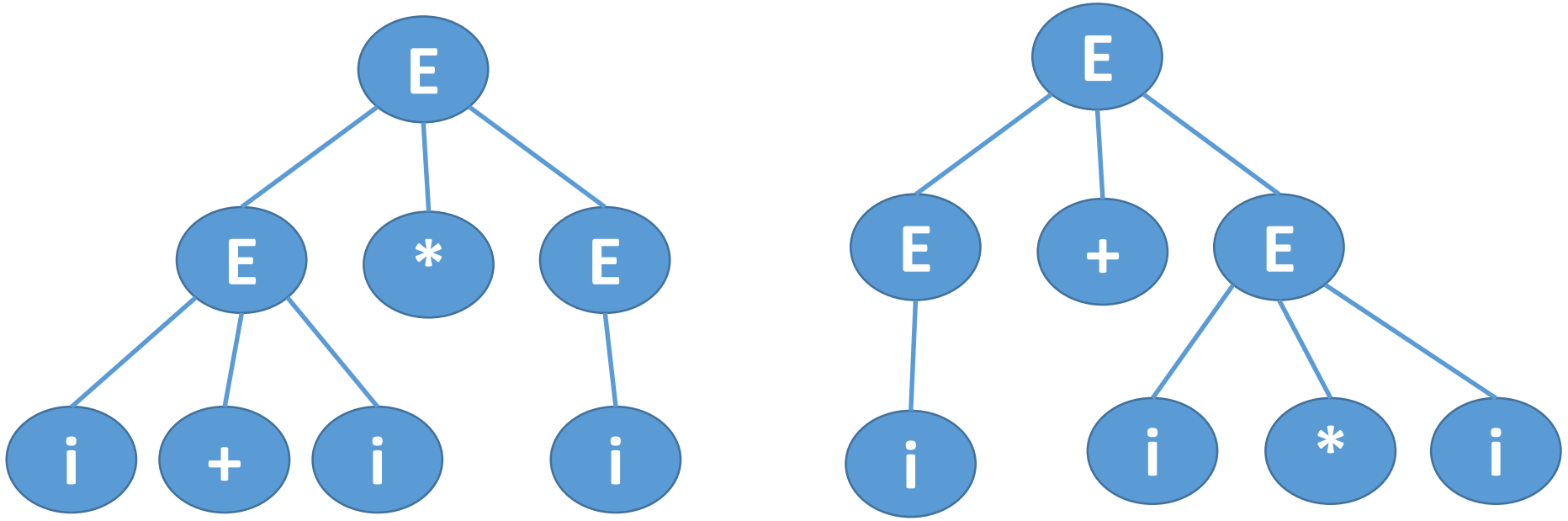
Цепочка до подстановки	Правило подстановки	Цепочка после подстановки	Дерево вывода
E			
E	$E \rightarrow E + T$	E + T	
E + T	$E \rightarrow T$	T + T	

Цепочка до подстановки	Правило подстановки	Цепочка после подстановки	Дерево вывода
T + T	$T \rightarrow P$	P + T	
…			
i + i * P	$P \rightarrow i$	i + i * i	

Пример

$$E \rightarrow E + E \mid E * E \mid i$$

i + i * i



Однозначность

Грамматика называется *однозначной*, если для каждой цепочки символов языка, заданного этой грамматикой, существует единственное дерево вывода

$$A \to AA \mid \alpha$$

$$A \to A\alpha A \mid \beta$$

$$A \to \alpha A \mid A\beta \mid \gamma$$

$$A \to \alpha A \mid \alpha A\beta A \mid \gamma$$

где $A \in \mathbf{VN}$; $\alpha, \beta, \gamma \in (\mathbf{VN} \cup \mathbf{VT})^*$

Регулярные множества

Все регулярные языки представляют собой регулярные множества

Регулярные множества – множества цепочек символов над заданным алфавитом, построенные определенным образом (с использованием операций объединения, конкатенации и итерации)

Регулярные выражения

Регулярные выражения – еще один способ определения регулярных языков

0 – регулярное выражение, обозначающее $\varnothing$

ε – регулярное выражение, обозначающее $\{\varepsilon\}$

a – регулярное выражение, обозначающее $\{a\}$ $\forall a \in \mathbf{V}$

Регулярные операции

- *Объединение* регулярных выражений E и F
  (E + F) – регулярное выражение, определяющее объединение языков **L**(E) и **L**(F)

- *Конкатенация* регулярных выражений E и F
  (EF) – регулярное выражение, определяющее конкатенацию языков **L**(E) и **L**(F)

- *Итерация* регулярного выражения E
  (E*) – регулярное выражение, определяющее итерацию языка **L**(E)

Приоритеты регулярных операторов

Оператор
итерация
конкатенация
объединение

$01^* + 1 = (0(1^*)) + 1$

$(01)^* + 1$

$0(1^* + 1)$

Задания для самостоятельной работы

Опишите языки, определяемые следующими регулярными выражениями:

1) $(1 + \varepsilon)(00^*1)^*0^*$
2) $(0^*1^*)^*000(0 + 1)^*$
3) $(0 + 10)^*1^*$

Регулярные выражения

- Два регулярных выражения, α и β, равны ($\alpha = \beta$), если они обозначают одно и то же множество

- Каждое регулярное выражение обозначает одно и только одно регулярное множество

- Для одного регулярного множества может существовать сколько угодно много регулярных выражений, обозначающих это множество

Свойства регулярных выражений

α, β, γ - регулярные выражения

1. $\alpha\alpha^* = \alpha^*\alpha = \alpha^+$
2. $\varepsilon + \alpha\alpha^* = \varepsilon + \alpha^*\alpha = \alpha^*$
3. $\alpha + \beta = \beta + \alpha$
4. $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
5. $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
6. $(\beta + \gamma)\alpha = \beta\alpha + \gamma\alpha$
7. $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
8. $\alpha + \alpha = \alpha$
9. $\alpha + \alpha^* = \alpha^*$
10. $\varepsilon + \alpha^* = \alpha^* + \varepsilon = \alpha^*$
11. $0^* = \varepsilon$
12. $0\alpha = \alpha 0 = 0$
13. $0 + \alpha = \alpha + 0 = \alpha$
14. $\varepsilon\alpha = \alpha\varepsilon = \alpha$
15. $(\alpha^*)^* = \alpha^*$

Регулярные выражения

- Для любого регулярного языка, заданного регулярной грамматикой, можно получить регулярное выражение, определяющее этот же язык

Имеется регулярная грамматика **G**(**VT**, **VN**, **P**, S), необходимо найти регулярное выражение над алфавитом **VT**, определяющее язык **L**(**G**), заданный этой грамматикой.

1. На основе грамматики **G** строим систему уравнений с регулярными коэффициентами
2. Решаем полученную систему уравнений. Решение, полученное для целевого символа грамматики, представляет собой искомое регулярное выражение

Система уравнений с регулярными коэффициентами

1. Обозначим нетерминальные символы следующим образом:
   VN = $\{X_1, X_2, \ldots, X_n\}$.
   Все правила будут иметь вид: $X_i \to \gamma X_j$ или $X_i \to X_j\gamma$, $X_j \in$ **VN**, $\gamma \in$ **VT***
2. Построим систему уравнений с регулярными коэффициентами

$$X_1 = \alpha_{01} + X_1\alpha_{11} + X_2\alpha_{21} + \ldots + X_n\alpha_{n1}$$

$$X_2 = \alpha_{02} + X_1\alpha_{12} + X_2\alpha_{22} + \ldots + X_n\alpha_{n2}$$

$$\ldots$$

$$X_n = \alpha_{0n} + X_1\alpha_{1n} + X_2\alpha_{2n} + \ldots + X_n\alpha_{nn}$$

Система уравнений с регулярными коэффициентами

Коэффициенты $\alpha_{01}, \alpha_{02}, \ldots, \alpha_{0n}$ выбираются следующим образом:
$\alpha_{0i} = (\gamma_1 + \gamma_2 + \ldots + \gamma_m)$, если есть правила вида $X_i \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$
$\alpha_{0i} = \varnothing$, если таких правил нет

Коэффициенты $\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jn}$ выбираются следующим образом:
$\alpha_{ji} = (\gamma_1 + \gamma_2 + \ldots + \gamma_m)$, если есть правила вида
$X_i \to X_j\gamma_1 \mid X_j\gamma_2 \mid \ldots \mid X_j\gamma_m$
$\alpha_{ji} = \varnothing$, если таких правил нет

3. Находим решение построенной системы уравнений

$X = X\alpha + \beta \Rightarrow \beta\alpha^*$ $\qquad$ $X = \alpha X + \beta \Rightarrow \alpha^*\beta$

Пример

$S \to A\perp \mid B\perp$

$A \to a \mid Ba$

$B \to b \mid Bb \mid Ab$

$\Rightarrow$

$S = X_1$

$A = X_2$

$B = X_3$

$\Rightarrow$

$X_1 \to X_2\perp \mid X_3\perp$

$X_2 \to a \mid X_3a$

$X_3 \to b \mid X_3b \mid X_2b$

Пример (продолжение)

$S \to A\perp \mid B\perp$

$A \to a \mid Ba$

$B \to b \mid Bb \mid Ab$

$S = X_1$

$A = X_2$

$B = X_3$

$X_1 \to X_2\perp \mid X_3\perp$

$X_2 \to a \mid X_3a$

$X_3 \to b \mid X_3b \mid X_2b$

$X_1 \to X_2\perp \mid X_3\perp$

$X_2 \to a \mid X_3a$

$X_3 \to b \mid X_3b \mid X_2b$

$X_1 \to a\perp \mid X_3a\perp \mid X_3\perp$

$X_2 \to a \mid X_3a$

$X_3 \to b \mid X_3b \mid ab \mid X_3ab$

Пример (продолжение)

$S \to A\perp \mid B\perp$

$A \to a \mid Ba$

$B \to b \mid Bb \mid Ab$

$S = X_1$

$A = X_2$

$B = X_3$

$X_1 = X_2\perp \mid X_3\perp$

$X_2 = a \mid X_3a$

$X_3 = b \mid X_3b \mid X_2b$

$X_1 = X_2\perp \mid X_3\perp$

$X_2 = a \mid X_3a$

$X_3 = b \mid X_3b \mid X_2b$

$X_1 = a\perp \mid X_3a\perp \mid X_3\perp$

$X_2 = a \mid X_3a$

$X_3 = b \mid X_3b \mid ab \mid X_3ab$

$X_3 = b \mid X_3b \mid ab \mid X_3ab$

$X_3 = X_3b \mid X_3ab \mid ab \mid b$

$X_3 = X_3(b \mid ab) \mid (ab \mid b)$

Пример (продолжение)

$$X_3 = X_3\underbrace{(b \mid ab)}_{\alpha} \mid \underbrace{(ab \mid b)}_{\beta}$$

$$\boxed{X = X\alpha + \beta \Rightarrow \beta\alpha^*}$$

Пример (продолжение)

$$X_3 = X_3\underbrace{(b \mid ab)}_{\alpha} \mid \underbrace{(ab \mid b)}_{\beta}$$

$$X = X\alpha + \beta \Rightarrow \beta\alpha^*$$

$$X_3 = (ab \mid b)(b \mid ab)^*$$

$$X_1 = a\bot \mid X_3a\bot \mid X_3\bot$$
$$X_2 = a \mid X_3a$$
$$X_3 = (ab \mid b)(b \mid ab)^*$$

$$\Rightarrow$$

$$X_1 = a\bot \mid (ab \mid b)(b \mid ab)^*\, a\bot \mid (ab \mid b)(b \mid ab)^*\bot$$
$$X_2 = a \mid (ab \mid b)(b \mid ab)^*a$$
$$X_3 = (ab \mid b)(b \mid ab)^*$$

Пример (продолжение)

X_1 = a⊥ | (ab | b)(b | ab)* a⊥ | (ab | b)(b | ab)*⊥

X_2 = a | (ab | b)(b | ab)*a

X_3 = (ab | b)(b | ab)*

Решением системы уравнений является регулярное выражение, полученное для целевого символа грамматики

S = a⊥ | (ab | b)(b | ab)* a⊥ | (ab | b)(b | ab)*⊥

A = a | (ab | b)(b | ab)*a

B = (ab | b)(b | ab)*

Пример (продолжение)

X_1 = a⊥ | (ab | b)(b | ab)* a⊥ | (ab | b)(b | ab)*⊥

X_2 = a | (ab | b)(b | ab)*a

X_3 = (ab | b)(b | ab)*

Решением системы уравнений является регулярное выражение, полученное для целевого символа грамматики

S = a⊥ | (ab | b)(b | ab)* a⊥ | (ab | b)(b | ab)*⊥

A = a | (ab | b)(b | ab)*a

B = (ab | b)(b | ab)*

$$L = \left\{ \begin{array}{c} a\perp \\ (ab \mid b)(b \mid ab)* \; a\perp \\ (ab \mid b)(b \mid ab)*\perp \end{array} \right\}$$

Литература

1. Молчанов А.Ю. Системное программное обеспечение: учебник для вузов. 3-е изд. – СПб.: Питер, 2010. – 400 с. (С. 104-111)
2. Хопкрофт Дж. Э., Мотвани Р., Ульман Дж. Д. Введение в теорию автоматов, языков и вычислений, 2-е изд. – М.: Вильямс, 2002. – 528 с. (раздел 3.1)
3. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции: Т. 1 Синтаксический анализ. – М.: Мир, 1978. – 613 с. (С. 124-133)

Курсовая работа

- *Задание* - создание транслятора с некоторого входного языка на заданный выходной язык.

- Формирование команд (не более 3 человек) для выполнения задания.

- Каждая команда разрабатывает свой входной язык программирования. В качестве выходного (результирующего) используется язык ассемблера.

Требования к входному языку

- Два простых типа данных
- Один оператор цикла (вложенный)
- Один оператор условного перехода (вложенный)
- Ввод данных с клавиатуры (read)
- Вывод данных на экран (write)

Курсовая работа

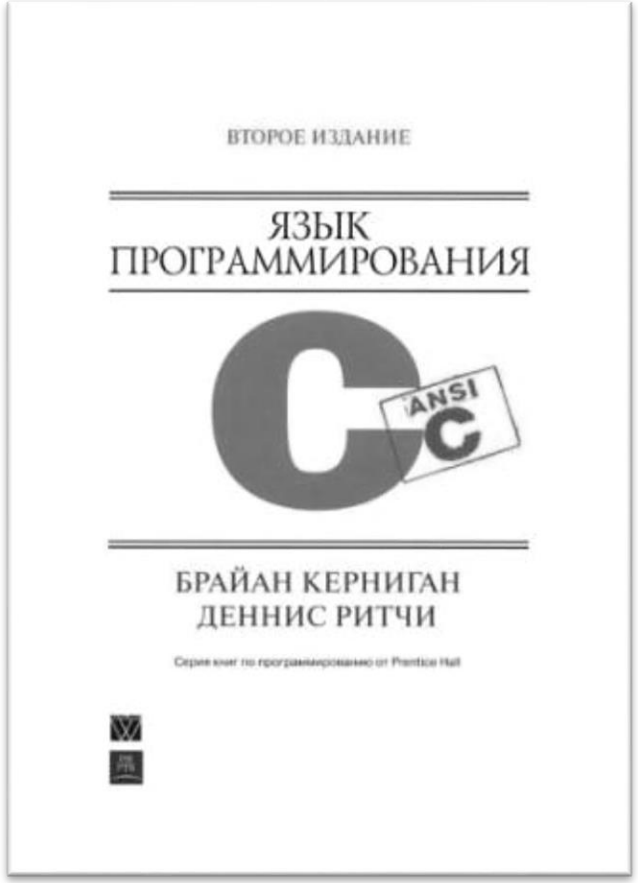
<u>Пояснительная записка должна содержать следующие разделы:</u>

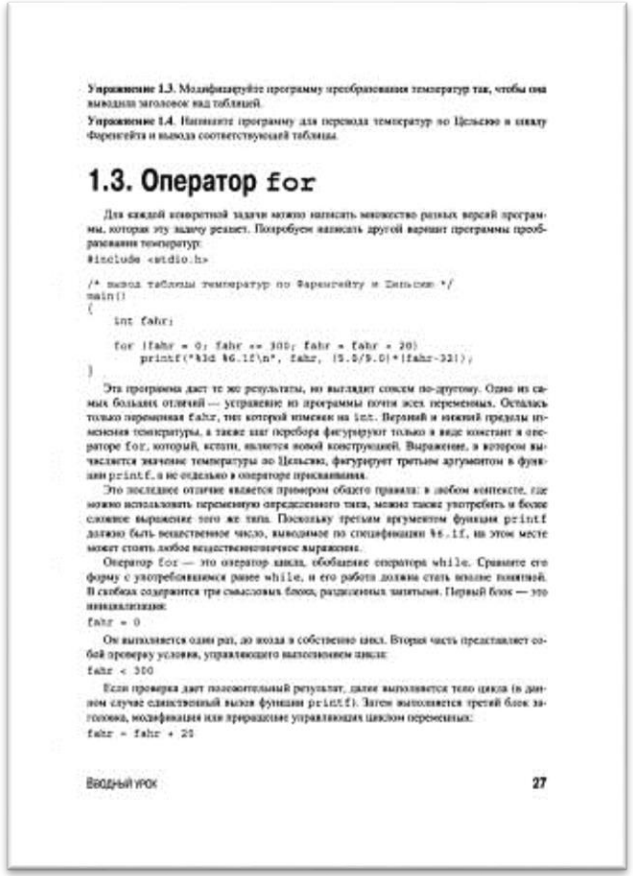
1. Введение (краткое изложение цели работы)
2. Учебное руководство по разработанному языку
3. Справочник по разработанному языку
4. Описание грамматики входного языка в одном из трех возможных видов:
   - форма Бэкуса-Наура;
   - расширенная форма Бэкуса-Наура;
   - графическая форма.
5. Структура разработанного транслятора
6. Описание средств разработки
7. Проблемы, возникшие при разработке транслятора и способы их решения
8. Примеры тестовых программ на входном языке и результирующих программ, сгенерированных транслятором
9. Текст программы транслятора (в приложении)

Курсовая работа

- *Учебное руководство* по разработанному языку должно содержать примеры нескольких программ, иллюстрирующих основные свойства и область применения разрабатываемого языка.

Керниган Б., Ричи Д. Язык программирования Си. – 2-е изд. – М.: Вильямс, 2009. – 304 с. (1 глава)





Упражнение 1.3. Модифицируйте программу преобразования температур так, чтобы она выводила заголовок над таблицей.

Упражнение 1.4. Напишите программу для перевода температур по Цельсию в шкалу Фаренгейта и вывода соответствующей таблицы.

1.3. Оператор `for`

Для каждой конкретной задачи можно написать множество разных версий программы, которая эту задачу решает. Попробуем написать другой вариант программы преобразования температур:

```
#include <stdio.h>

/* вывод таблицы температур по Фаренгейту и Цельсию */
main()
{
    int fahr;

    for (fahr = 0; fahr <= 300; fahr = fahr + 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

Эта программа дает те же результаты, но выглядит совсем по-другому. Одно из самых больших отличий — устранение из программы почти всех переменных. Осталась только переменная `fahr`, тип которой изменен на `int`. Верхний и нижний пределы изменения температуры, а также шаг перебора фигурируют только в виде констант в операторе `for`, который, кстати, является новой конструкцией. Выражение, в котором вычисляется значение температуры по Цельсию, фигурирует третьим аргументом в функции `printf`, а не отдельно в операторе присваивания.

Это последнее отличие является примером общего правила: в любом контексте, где можно использовать переменную определенного типа, можно также употребить и более сложное выражение того же типа. Поскольку третьим аргументом функции `printf` должно быть вещественное число, выводимое по спецификации `%6.1f`, на этом месте может стоять любое вещественнозначное выражение.

Оператор `for` — это оператор цикла, обобщение оператора `while`. Сравните его форму с употребляемыми ранее `while`, и его работа должна стать ясным понятной. В скобках содержится три смысловых блока, разделенных запятыми. Первый блок — это инициализация:

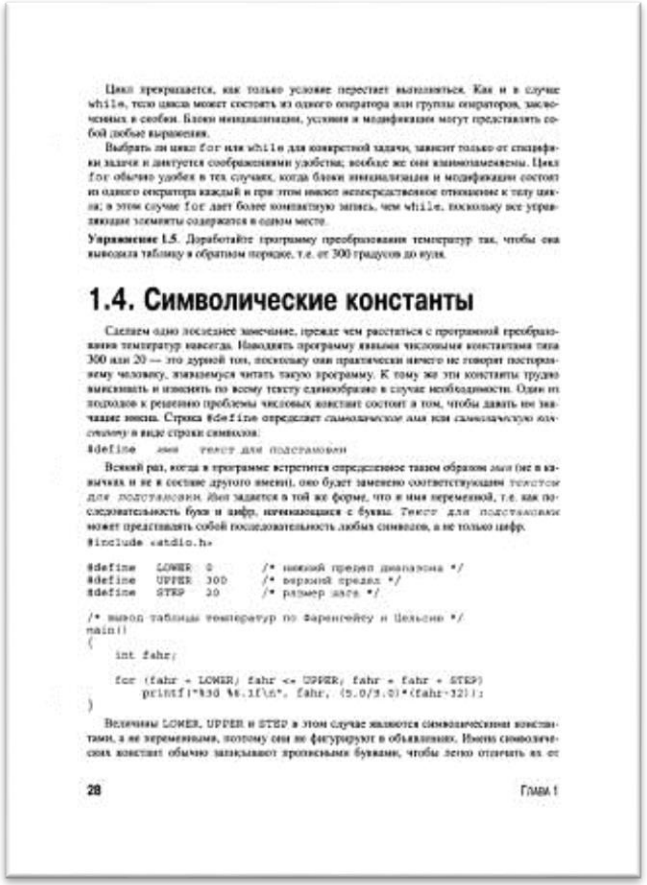
```
fahr = 0
```

Он выполняется один раз, до входа в собственно цикл. Вторая часть представляет собой проверку условия, управляющего выполнением цикла:

```
fahr < 300
```

Если проверка дает положительный результат, далее выполняется тело цикла (в данном случае единственный вызов функции `printf`). Затем выполняется третий блок заголовка, модификация или приращение управляющих циклом переменных:

```
fahr = fahr + 20
```

Цикл прекращается, как только условие перестает выполняться. Как и в случае `while`, тело цикла может состоять из одного оператора или группы операторов, заключенных в скобки. Блоки инициализации, условия и модификации могут представлять собой любые выражения.

Выбрать ли цикл `for` или `while` для конкретной задачи, зависит только от специфики задачи и диктуется соображениями удобства; вообще же они взаимозаменяемы. Цикл `for` обычно удобен в тех случаях, когда блоки инициализации и модификации состоят из одного оператора каждый и при этом имеют непосредственное отношение к телу цикла; в этом случае `for` дает более компактную запись, чем `while`, поскольку все управляющие элементы содержатся в одном месте.

Упражнение 1.5. Доработайте программу преобразования температур так, чтобы она выводила таблицу в обратном порядке, т.е. от 300 градусов до нуля.

1.4. Символические константы

Сделаем одно последнее замечание, прежде чем расстаться с программой преобразования температур навсегда. Наводить программу явными числовыми константами типа 300 или 20 — это дурной тон, поскольку они практически ничего не говорят постороннему человеку, взявшемуся читать такую программу. К тому же эти константы трудно выискивать и изменить по всему тексту единообразно в случае необходимости. Один из подходов к решению проблемы числовых констант состоит в том, чтобы давать им значащие имена. Строка `#define` определяет *символическое имя* или *символическую константу* в виде строки символов:

```
#define   имя   текст для подстановки
```

Всякий раз, когда в программе встретится определенное таким образом *имя* (не в кавычках и не в составе другого имени), оно будет заменено соответствующим *текстом для подстановки*. *Имя* задается в той же форме, что и имя переменной, т.е. как последовательность букв и цифр, начинающаяся с буквы. *Текст для подстановки* может представлять собой последовательность любых символов, а не только цифр.

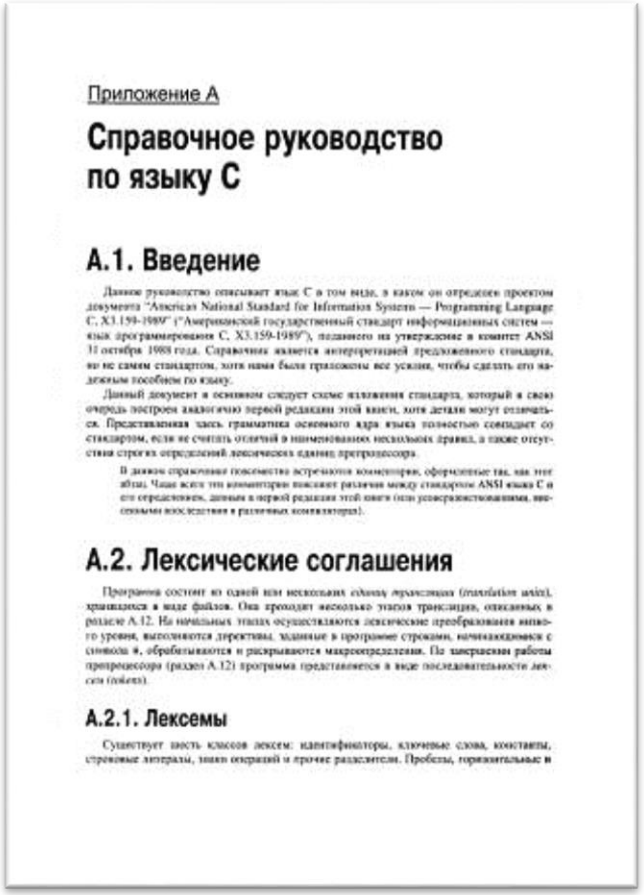
```
#include <stdio.h>

#define   LOWER  0      /* нижний предел диапазона */
#define   UPPER  300    /* верхний предел */
#define   STEP   20     /* размер шага */

/* вывод таблицы температур по Фаренгейту и Цельсию */
main()
{
    int fahr;

    for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

Величины `LOWER`, `UPPER` и `STEP` в этом случае являются символическими константами, а не переменными, поэтому они не фигурируют в объявлениях. Имена символических констант обычно записывают прописными буквами, чтобы легко отличать их от

Курсовая работа

- *Справочник по языку* должен содержать полное описание его лексической и синтаксической структуры, включая полную грамматику языка

Керниган Б., Ричи Д. Язык программирования Си. – 2-е изд. – М.: Вильямс, 2009. – 304 с. (Приложение А)



Приложение А

Справочное руководство по языку С

А.1. Введение

Данное руководство описывает язык С в том виде, в каком он определен проектом документа "American National Standard for Information Systems — Programming Language C, X3.159-1989" ("Американский государственный стандарт информационных систем — язык программирования С, X3.159-1989"), поданного на утверждение в комитет ANSI 31 октября 1988 года. Справочник является интерпретацией предложенного стандарта, но не самим стандартом, хотя нами были приложены все усилия, чтобы сделать его надежным пособием по языку.

Данный документ в основном следует схеме изложения стандарта, который в свою очередь построен аналогично первой редакции этой книги, хотя детали могут отличаться. Представленная здесь грамматика основного ядра языка полностью совпадает со стандартом, если не считать отличий в наименованиях нескольких правил, а также отсутствия строгих определений лексических единиц препроцессора.

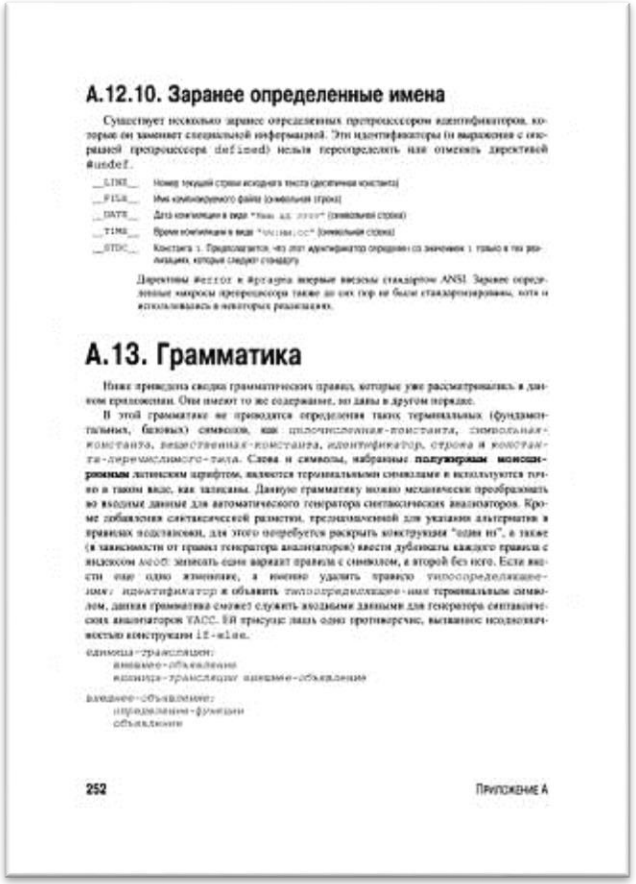
В данном справочнике повсеместно встречаются комментарии, оформленные так, как этот абзац. Чаще всего эти комментарии поясняют различия между стандартом ANSI языка С и его определением, данным в первой редакции этой книги (или усовершенствованиями, внесенными впоследствии в различных компиляторах).

А.2. Лексические соглашения

Программа состоит из одной или нескольких *единиц трансляции* (*translation units*), хранящихся в виде файлов. Она проходит несколько этапов трансляции, описанных в разделе А.12. На начальных этапах осуществляются лексические преобразования низкого уровня, выполняются директивы, заданные в программе строками, начинающимися с символа #, обрабатываются и раскрываются макроопределения. По завершении работы препроцессора (раздел А.12) программа представляется в виде последовательности *лексем* (*tokens*).

А.2.1. Лексемы

Существует шесть классов лексем: идентификаторы, ключевые слова, константы, строковые литералы, знаки операций и прочие разделители. Пробелы, горизонтальные и



А.12.10. Заранее определенные имена

Существует несколько заранее определенных препроцессором идентификаторов, которые он заменяет специальной информацией. Эти идентификаторы (и выражения с операцией препроцессора defined) нельзя переопределять или отменять директивой #undef.

__LINE__	Номер текущей строки исходного текста (десятичная константа)
__FILE__	Имя компилируемого файла (символьная строка)
__DATE__	Дата компиляции в виде "Mmm dd yyyy" (символьная строка)
__TIME__	Время компиляции в виде "hh:mm:ss" (символьная строка)
__STDC__	Константа 1. Предполагается, что этот идентификатор определен со значением 1 только в тех реализациях, которые следуют стандарту

Директивы #error и #pragma вскоре внесены стандартом ANSI. Заранее определенные макросы препроцессора также до сих пор не были стандартизированы, хотя и использовались в некоторых реализациях.

А.13. Грамматика

Ниже приведена сводка грамматических правил, которые уже рассматривались в данном приложении. Они имеют то же содержание, но даны в другом порядке.

В этой грамматике не приводится определения таких терминальных (фундаментальных, базовых) символов, как *целочисленная-константа*, *символьная-константа*, *вещественная-константа*, *идентификатор*, *строка* и *константа-перечислимого-типа*. Слова и символы, набранные **полужирным моноширинным** латинским шрифтом, являются терминальными символами и используются точно в таком виде, как записаны. Данную грамматику можно механически преобразовать во входные данные для автоматического генератора синтаксических анализаторов. Кроме добавления синтаксической разметки, предназначенной для указания альтернатив в правых подстановках, для этого потребуется раскрыть конструкции "один из", а также (в зависимости от правил генератора анализаторов) ввести дубликаты каждого правила с ядром *необ*: записать один вариант правила с символом, а второй без него. Если ввести еще одно изменение, а именно удалить правило *типоопределяющее-имя*: *идентификатор* и объявить *типоопределяющее-имя* терминальным символом, данная грамматика сможет служить входными данными для генератора синтаксических анализаторов YACC. Ей присущ лишь один противоречие, вызванное неоднозначностью конструкции if-else.

```
единица-трансляции:
    внешнее-объявление
    единица-трансляции внешнее-объявление

внешнее-объявление:
    определение-функции
    объявление
```


```
определение-функции:
    спецификаторы-объявления_опц описатель список-объявлений_опц со-
ставной-оператор

объявление:
    спецификаторы-объявления список-инициализ-описателей_опц

список-объявлений:
    объявление
    список-объявлений объявление

спецификаторы-объявления:
    спецификатор-класса-памяти спецификаторы-объявления_опц
    спецификатор-типа спецификаторы-объявления_опц
    модификатор-типа спецификаторы-объявления_опц

спецификатор-класса-памяти: один из
    auto    register    static    extern    typedef

спецификатор-типа: один из
    void char short int long float double signed
    unsigned спецификатор-структуры-или-объединения
    спецификатор-перечисления типоопределяющее-имя

модификатор-типа: один из
    const    volatile

спецификатор-структуры-или-объединения:
    struct-или-union идентификатор_опц { список-объявлений-структуры }
    struct-или-union идентификатор

struct-или-union: один из
    struct    union

список-объявлений-структуры:
    объявление-структуры
    список-объявлений-структуры объявление-структуры

список-инициализ-описателей:
    инициализ-описатель
    список-инициализ-описателей , инициализ-описатель

инициализ-описатель:
    описатель
    описатель = инициализатор

объявление-структуры:
    список-спецификаторов-модификаторов список-описателей-структуры ;

список-спецификаторов-модификаторов:
    спецификатор-типа список-спецификаторов-модификаторов_опц
    модификатор-типа список-спецификаторов-модификаторов_опц

список-описателей-структуры:
    описатель-структуры
    список-описателей-структуры , описатель-структуры
```

Курсовая работа

- Транслятор должен запускаться из командной строки с несколькими входными параметрами. Первым и главным входным параметром должно быть имя файла программы на входном языке, вторым параметром может быть имя результирующего файла. Требования к остальным параметрам командной строки и управляющим ключам (если они необходимы) устанавливаются самостоятельно. Командная строка должна быть достаточной для функционирования транслятора. Помимо интерфейса командной строки возможно наличие дополнительного интерактивного интерфейса пользователя у транслятора (в том числе и графического) по усмотрению исполнителя работы.

- Описание входного языка разрабатывается исполнителем самостоятельно. Входная программа может быть разбита на строки произвольным образом, все пробелы и переводы строки должны игнорироваться транслятором. Текст входной программы может содержать комментарии любой длины, которые должны игнорироваться транслятором (вид комментария задается исполнителем).

Курсовая работа

- В качестве выходного (результирующего) используется язык ассемблера.

- В случае если на вход транслятора подается входная программа, содержащая лексические, синтаксические или семантические ошибки, транслятор должен корректно завершать свое выполнение и выдавать сообщение о найденной ошибке во входной программе с указанием строки, в которой найдена ошибка. По возможности транслятор должен указывать тип найденной ошибки. Транслятор должен указать несколько ошибок во входной программе, если они были им обнаружены.

Курсовая работа

Результатами выполнения курсовой работы являются программная реализация транслятора и пояснительная записка, оформленная в соответствии с требованиями и заданием на работу.