ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

3 лабы, уже есть в ЛК (или где-то там). Для третьей уже есть файлы с вариантами (она объемная), на первую и вторую вариантов нет, делаем все задания. На практиках решаем задачки, за выходы к доске дают баллы, на последней практике контрольная с теми же задачами по типу, доп баллы будут учитываться только в счет автомата по контрольной. Максимум за контрольную 20 баллов, из них 10 максимум дополнительных. Всего можно набрать 95 баллов, набираешь 90 и контрольная минимум 10 баллов – пятерка, на остальные оценки: на 4 больше 80, на 3 больше 65, контрольная не учитывается. На лабы желательно приходить с готовыми лабами.

Дальше впадлу писать было сори

ЛЕКЦИЯ 1

3 группы ЯП:

1) Машинный – набор команд, который интерпретируется на аппаратном уровне
2) Языки Assembler – языки низкого уровня – отображают команды некоторой машины

3) Языки высокого уровня – имеют сложную структуру, не зависят от ОС и набора команд.

Транслятор – обслуживающая программа, преобразующая исходную программу, представленную на входном ЯП, в рабочую программу, представленную на объектном языке.

Основные виды трансляторов:

- ***Assembler***
- ***Компилятор***
- **Интерпретатор**

Их модификации

- Эмулятор
- Перекодировщик
- Макропроцессор

Ассемблер – системно обслуживающая программа, которая преобразует символические конструкции в команды машинного языка. Ассемблер предназначен для лучшего восприятия машинных команд. Заменяет 1 машинный код 1 командой

Компилятор – обслуживающая программа, выполняющая трансляцию на машинный язык программы, написанной на исходном языке программирования. Компилятор может заменять до 100 машинных команд одной командой.

Интерпретатор - программа или устройство, осуществляющая пооператорную трансляцию и выполнение исходной программы. Не порождает программы на машинном языке, программа выполняется в 50-100 раз медленнее чем на машинных кодах.

Любой язык программирования имеет

- **Алфавит**
- **Лексику**
- **Операторы**
- **Семантику**
- **Прагматизм**
- **Семиотику**

Фазы трансляции

1. **Лексический анализ**
2. **Синтаксический анализ**
3. **Семантический анализ**
4. **Синтез объектной программы**
5. **Генератор кода**
6. **Анализатор ошибок**

Лексический анализ – перевод исходной программы на внутренний язык компилятора (выделение лексем)

Синтаксический анализ – переводит последовательность образов лексем в форму промежуточной программы, которая более четко отражает структуру исходной (т.е. порядок и связь между операторами)

Семантический анализ – проверка семантических соглашений в программе

Синтез объектной программы – начинается с выделения и распределения памяти для основных объектов, анализируется каждое предложение исходной программы и генерируется семантически эквивалентное предложение объектного языка

Генератор кода - развертывает атомы, построенные синтаксическим анализатором в последовательность команд, которые выполняют соответствующие действия

Анализатор ошибок - получает информацию об ошибках, формирует сообщение пользователей и выполняет действия, связанные с корректным завершением работы программы в случае, когда дальнейшая трансляция невозможна.

Многопроходная организация - при ней каждая фаза является независимым процессом, передающем управление следующей фазе только после окончания обработки своих данных.

Однопроходная организация – при ней все фазы представляют единый процесс и передают друг другу данные небольшими фрагментами.

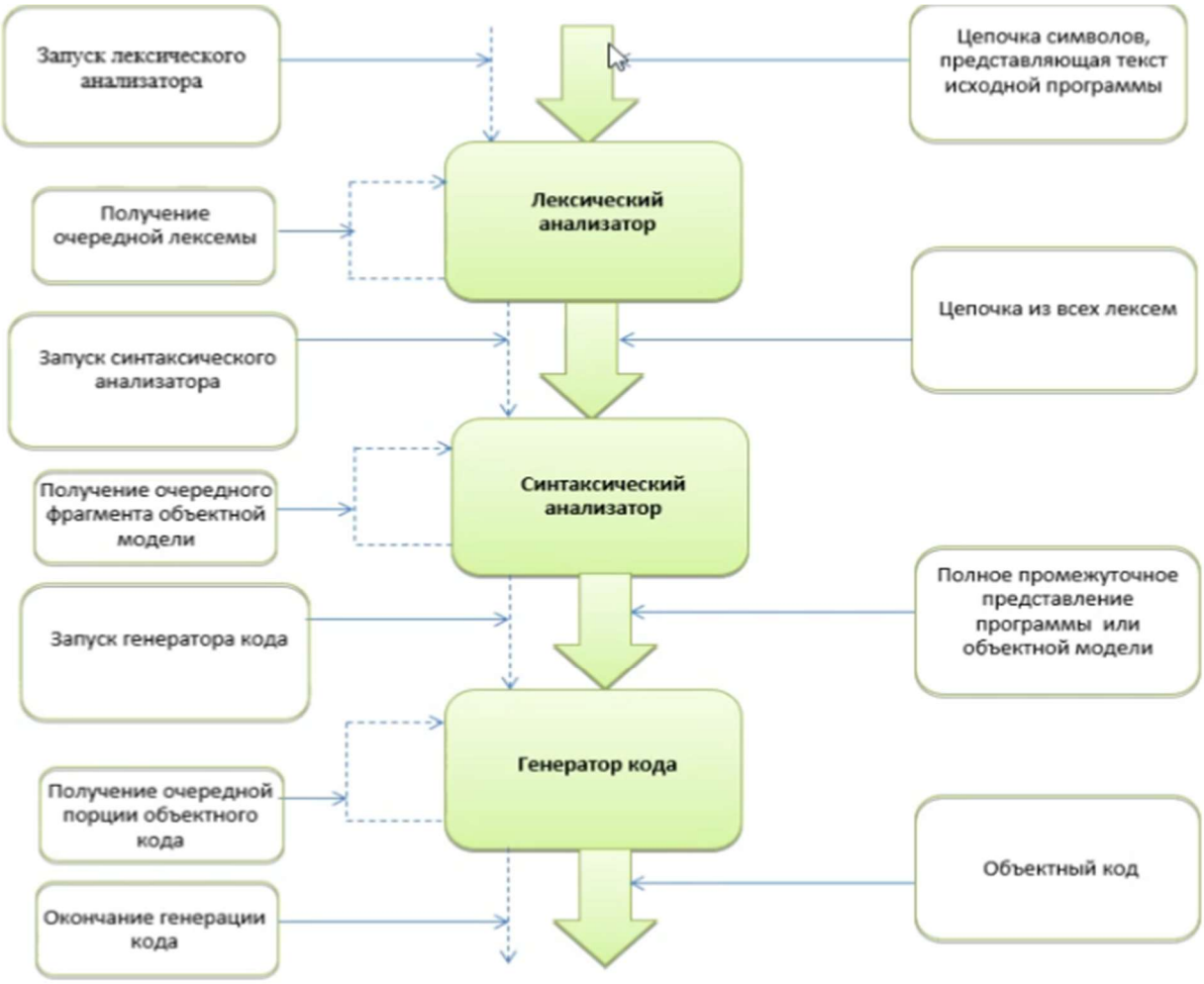
Варианты взаимодействия блоков транслятора

Выделяют два основных варианта взаимодействия:

- Многопроходная организация
- Однопроходная организация

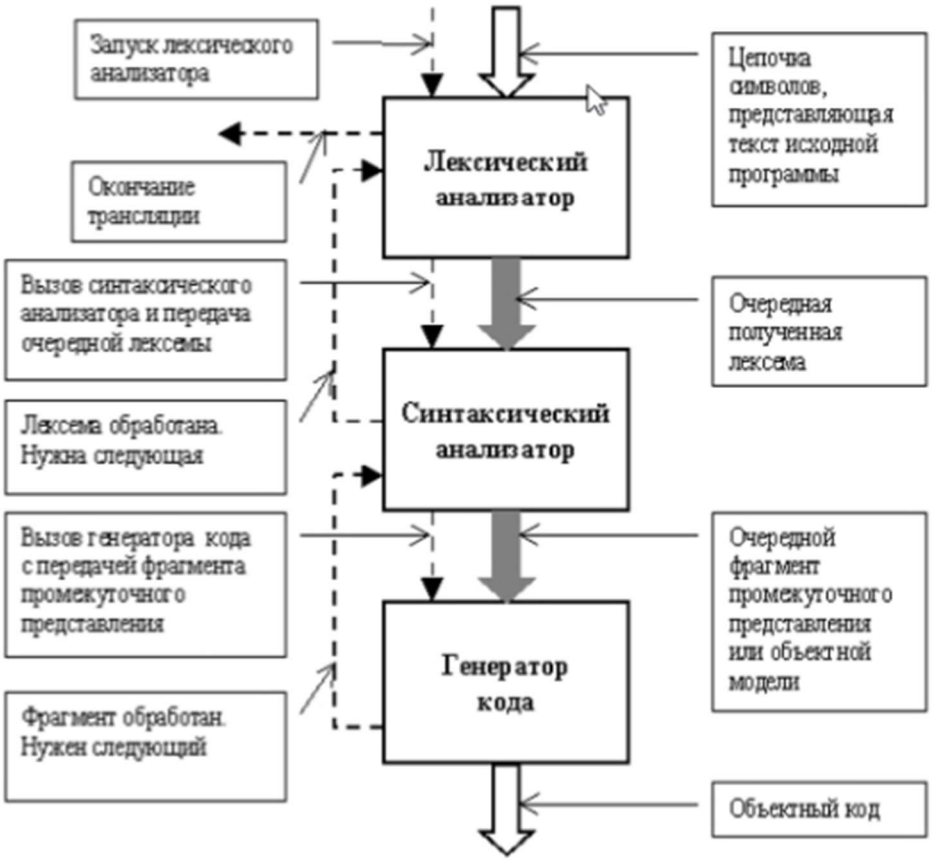
На основе двух основных вариантов возможны различные сочетания.

Многопроходная организация на примере компилятора: 



2.Однопроходное взаимодействие

а) При управлении лексическим анализатором



Формальный язык – множество цепочек конченой длины в алфавите А, задаваемое некоторым множество правил

Механизм порождения позволяет описывать языки с помощью системы правил, называемых грамматикой.

Замыкание алфавита (итерация) – множество всех возможных цепочек над алфавитом.

Формальные языки и грамматики

- Пусть А-алфавит

  $\mathbf{A_n}$=A×A×….A

  $a_1a_2a_3a_4…..a_n$ –цепочки

  $(a_1 , a_2 , a_3 , a_4 ,…..a_n)$ – так нельзя
- При **n=1**, буквы также являются цепочками.
- Цепочка, не имеющая букв, называется **пустой**, обозначается **λ**, не принадлежит алфавиту.
- **Замыкание** или **итерация** алфавита А, обозначают А*:

$$A^* = A^0 \bigcup A^1 \bigcup \ldots = \bigcup_{n=0}^{\infty} A^n \text{ ,где } \quad A^0 = \{\lambda\},\ A^1 = A$$

(число нулей всегда равно числу единиц, если одинаковый индекс)

Формальные языки и грамматики

Использование системы множеств:

$L= \{0^n1^n|n\geq0\}$

Примеры синтаксисов:

1) $\{a^nb^nc^n|n\geq0\}$
2) $\{a^mb^n|\ m,n\geq0\}$
3) $\{x^n|n$ - простое число$\}$
4) $\{\ a^mb^nc^p\ |m=n$ или $n=p\}$
5) $\{\ a^nb^na^n\ |n>0\}$

Примеры (картинка выше):
1) aaabbbccc, n = 3;

2) aaabb, m = 3, n = 2;

3) xxxxx, n = 5;

4) aaabbcc, m = 3, n = 2, p = 2;

5) aabbaa, n = 2;

Формальные языки и грамматики

- Множество непустых цепочек над алфавитом A называется **усеченной итерацией**:

$$A^+ = \bigcup_{n=1}^{\infty} A^n = A^* \setminus \{\lambda\}$$

Каждая цепочка $\alpha \in A^*$ имеет конечную длину $|\alpha|$, при чем $|\lambda| = 0$

- Пусть $\alpha, \beta \in A$ принадлежат алфавиту. Цепочка β называется **подцепочкой** α, если $\alpha = \gamma\beta\delta$, причем $\gamma, \delta \in A^*$

(если справа и слева от беты есть символы, причем принадлежащие алфавиту A, если над A снежинки (*) – туда могут входить пустые цепочки, если (+)– любые кроме пустых)

- **Язык L над алфавитом A** – это множество цепочек итераций алфавита A^*
- Получается $L \subseteq A^*$, отсюда:

$$L^0 = \{\lambda\}$$

$$L^n = L^{n-1} \cdot L, \quad n \in N$$

$$L^+ = \bigcup_{n=1}^{\infty} L^n$$

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

Формальной порождающей грамматикой G называется четверка G=<N,T,P,S>,

где $V=T \cup N$, $T \cap N=\{\varnothing\square$

$S \in N$

P – продукции вида $\alpha \rightarrow \beta$,

где

$$\alpha \in (T \cup N)^* \cdot N(T \cup N)^* = V^+$$

$$\beta \in (T \cup N)^* = V^*$$

Терминалы a,b,c….., либо 0,1….9.
Нетерминалы: A, B, C, D, S.
Цепочки произвольной длины: α, β, γ, δ
Пустая цепочка: λ

N – конечное непустое множество **нетерминальных** символов

T – конечное непустое множество **терминальных** символов
S – символ из множества N, называемый начальным
P – конечное множество правил (продукций) грамматики

Вывод – последовательность подстановки.

Пример: Грамматикой, генерирующей язык, задаваемый синтаксисом $L=\{0^n1^n \mid n \geq 0\}$, является:

$G=\{N,T,P,S\}$, $N=\{S\}$, $T=\{0,1\}$, $S=\{S\}$,

$P=\{$ 1. $S \rightarrow 0S1$

2. $S \rightarrow \lambda\}$

Выводы цепочек:

$S \overset{1}{\rightarrow} 0S1 \overset{1}{\rightarrow} 00S11 \overset{2}{\rightarrow} 0011$ (n=2)

$S \overset{2}{\rightarrow} \lambda$ (n=0)

Пример: G={N,T,P,S}

T={дом, дуб, старый, заслоняет}

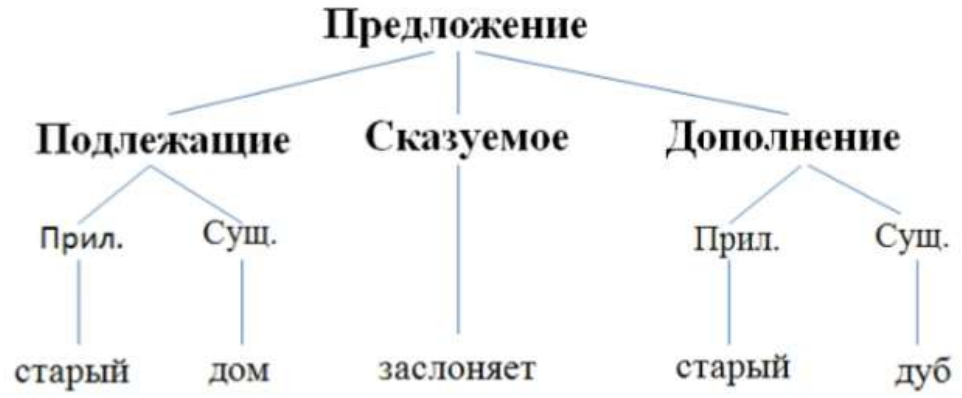
N ={Предложение, Подлежащие, Сказуемое,
Дополнение, Прилагательное, Существительное}

S={Предложение}

P={1. Предложение→ Подлежащие Сказуемое Дополнение
2. Подлежащие → Прилагательное Существительное
3. Дополнение → Прилагательное Существительное
4.Сказуемое → заслоняет
5.Прилагательное → старый
6.Существительное → дом
7.Существительное → дуб}

Вывод цепочки:

Предлож$\xrightarrow{1}$Подл. Сказ. Допол.$\xrightarrow{2}$Прил. Сущ. Сказ. Допол.$\xrightarrow{3}$Прил. Сущ. Сказ. Прил. Сущ.$\xrightarrow{5}$ старый Сущ. Сказ. старый Сущ.$\xrightarrow{6,7}$старый дом Сказ. старый дуб $\xrightarrow{4}$ старый дом заслоняет старый дуб



Рассмотрим грамматику G=({a,b,c,}, {S,A,B}, P, S)

P={

1. S→aABc
2. S→λ
3. A→cSB
4. A→Ab
5. B→bB
6. B→a}

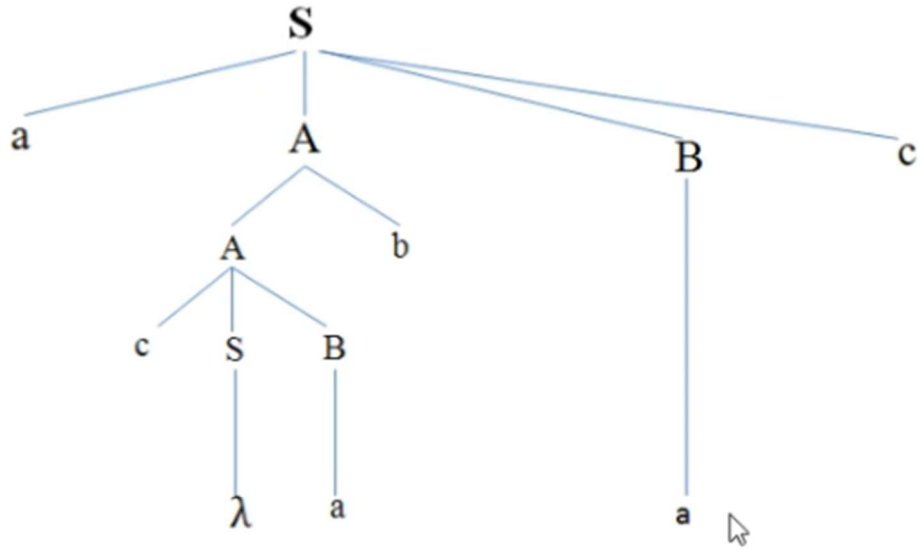
Пусть дана цепочка aABc, тогда цепочка acabac может иметь след. вывод:

$$S \xrightarrow{1} aABc \xrightarrow{4} aAbBc \xrightarrow{3} acSBbBc \xrightarrow{2} acBbBc \xrightarrow{6} acabBc \xrightarrow{6} acabac$$

$$S \Rightarrow^* acabac$$

(если мы поставили *, значит мы когда-то смогли вывести упоминаемую цепочку, удобно, когда делается много однотипных действий с одинаковыми шагами)

Дерево вывода терминальной цепочки acabac



Левый или левосторонний вывод $\alpha \rightarrow_L \beta$

Правый или правосторонний вывод $\alpha \rightarrow_R \beta$

Классификация грамматик и языков по Хомскому

0 – класс: правила вывода P грамматики G имеют форму $\varphi \rightarrow \omega$, без каких-либо ограничений на строки $\alpha \rightarrow \beta$.

1 – класс: все элементы множества P имеют форму $\varphi \rightarrow \omega$,

где $\varphi = \varepsilon_1 \alpha \varepsilon_2$, $\omega = \varepsilon_1 \beta \varepsilon_2$; $\varepsilon_1, \varepsilon_2 \in V^*$, $\alpha \in N$, $\beta \in V^+$.

контекстно-зависимая грамматика (КЗ)

если ($|\varphi| \leq |\omega|$), то **не укорачивающая грамматика**

Иерархия языков, грамматик и автоматов

3) Контекстно-зависимая грамматика (класс 1)

G4 = { {B,C,S}, {a,b,c}, P, S}

P:

1. S→aSBC
2. S→abc
3. cB→Bc
4. bB→bb
5. bC→bc
6. cC→cc

$S^1 \rightarrow aSBC^2 \rightarrow aabcBC^3 \rightarrow aabBcC^{4,6} \rightarrow aabbcc$

$L(G) = \{ a^n b^n c^n, n \geq 1\}$

2 – класс: все порождающие правила имеют вид $A \to \beta$, где $A \in N$, $\beta \in V^+$.

контекстно-свободная грамматика (КС)

если $\beta \in V^*$, то **укорачивающая КС- грамматика**.

3 – класс: все порождающие правила имеют вид $A \to bB$, $A \to b$, где A и $B \in N$, $b \in T$

автоматные или А-грамматики

Теорема №1:L(G) легко распознаваем

Теорема №2: $L(G)\backslash\lambda$ является языком класса 0

2) Контекстно-свободная грамматика (класс 2)

$G3 = \{ \{E,T,F\}, \{a,+,*,(,)\}, P, E\}$

P:

1. $E \to T$
2. $E \to E+T$
3. $T \to F$
4. $T \to T*F$
5. $F \to (E)$
6. $F \to a$

$E^2 \to E+T^1 \to T+T^3 \to F+F^6 \to a+a$

L(G) = {арифметические выражения}

Иерархия языков, грамматик и автоматов

Способы записи синтаксиса языка

1. Метаязык Хомского

- → отделяет левую часть правила от правой.
- Нетерминалы обозначаются буквами A_i ,где i – номер нетерминала.
- Терминалы – это символы, используемые в описываемом языке.
- Описание идентификатора:

1. $A_1 \rightarrow A$ 52. $A_1 \rightarrow z$
2. $A_1 \rightarrow B$ 53. $A_2 \rightarrow 0$

…… ……..

26. $A_1 \rightarrow Z$ 62. $A_2 \rightarrow 9$ 64. $A_3 \rightarrow A_3A_1$
27. $A_1 \rightarrow a$ 63. $A_3 \rightarrow A_1$ 65. $A_3 \rightarrow A_3A_2$

2. Метаязык Хомского-Щутценберже

- = отделяет левую часть правила от правой.
- Нетерминалы обозначаются буквами Ai ,где i - номер нетерминала.
- Терминалы – это символы, используемые в описываемом языке.
- Описание идентификатора:

1. $A_1 = A + B + \ldots + Z + a + \ldots + z$
2. $A_2 = 0 + 1 + 2 + \ldots + 9$
3. $A_3 = A_1 + A_3A_1 + A_3A_2$

3. Формы Бэкуса-Наура (БНФ)

- ::= отделяет левую часть правила от правой.
- Нетерминалы обозначаются произвольной символьной строкой, заключенной в < >.
- Терминалы – это символы, используемые в описываемом языке.
- Описание идентификатора:

1. <буква> ::= A | B | C | … | Z | a | … | z
2. <цифра> ::= 0 | 1 | 2 | … | 8 | 9
3. <идентификатор> ::= <буква> | <идентификатор> <буква> | <идентификатор> <цифра>

4. Расширенные форма Бэкуса-Наура (РБНФ)

- **[ ]** синтаксическая конструкция может отсутствовать
- **{ }** повторение (возможно от 0 раз)
- **( )** ограничение альтернативных конструкций
- **{/ /}** повторение 1 и более раз.
- **Нетерминалы** – изображаются словами, на русском языке.
- **Терминалы** – это слова, написанные буквами латинского алфавита или цепочками знаков, заключенных в кавычки.

- $ ставится в начале правила.
- Каждое правило заканчивается точкой.
- = отделяет левую часть правила от правой.
- | альтернативы.
- Если нетерминал состоит из нескольких слов, то они пишутся слитно.
- Описание идентификатора:

$буква =
"A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|"N"|
"O"|"P"|"Q"|"R«|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z".

$цифра = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9".

$идентификатор = буква { буква | цифра }.