Лекция 17

БГТУ, ФИТ, ПОИТ, 3 семестр, Языки программирования

Итоги предыдущего семестра

1. **Система программирования**: трансляторы, компоновщики, отладчики, профилировщики, программные библиотеки, IDE.

2. **Язык программирования**: формальная знаковая система, предназначенная для записи компьютерных программ. Знаковая система определяет набор лексических, синтаксических и семантических правил написания программы (программного кода). Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

3. **Исходный код (исходная программа)**: текст программы, написанный на языке программирования.

4. **Транслятор:** программа, преобразующий исходный код на одном языке программирования в исходный код на другом языке – целевом языке.

5. **Язык ассемблера** – машинно-ориентированный язык программирования (тесно связанный с архитектурой процессора). Язык низкого уровня, с помощью которого программист получает прямой доступ к аппаратным ресурсам компьютера (требуются познания в архитектуре процессоров определенного семейства и структуре ОС). Язык ассемблера не переносим по определению.
   Ассемблер - транслятор с исходного кода на языке ассемблера в программу на машинном языке (язык, который может интерпретироваться процессором и определяет выполняемые им действия).

6. **Объектный код:** результат работы транслятора. Один файл объектного кода – **объектный модуль.** Объектный модуль — двоичный файл, который может быть объединён с другими объектными файлами при помощи редактора связей (компоновщика) для получения готового исполняемого модуля, либо библиотеки.

7. **Компоновщик (linker, редактор связей):** программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль. Компоновщик выполняет разрешение внешних адресов памяти, по которым код из одного файла может обращаться к информации из другого файла.

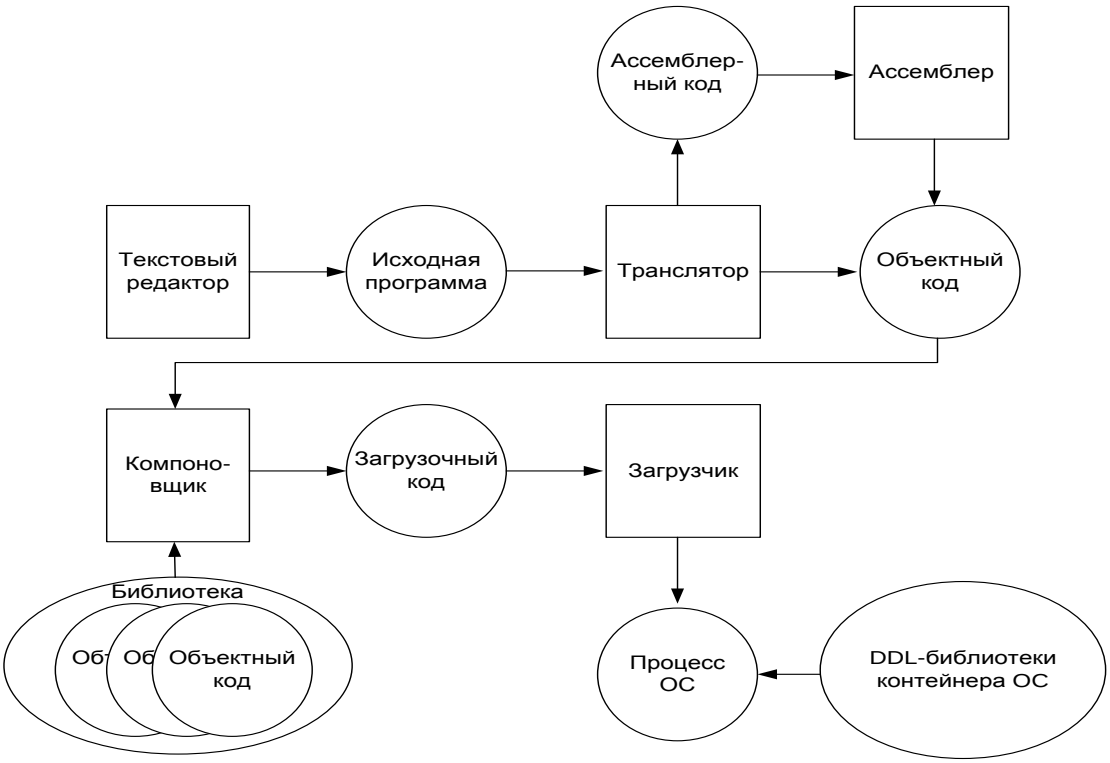
   Основные задачи редактора связей:

   - связывание между собой по внешним данным объектных модулей, порождаемых компилятором и составляющих единую программу;
   - подготовка таблицы трансляции относительных адресов для загрузчика;
   - статическое подключение библиотек с целью получения единого исполняемого модуля;
   - подготовка таблицы точек вызова функций динамических библиотек.

8. **Загрузочный код:** результат работы компоновщика. Один файл загрузочного кода – **загрузочный модуль.** Загрузочный модуль (или исполняемый файл) – файл, который может быть запущен на выполнение процессором под управлением операционной системы.

9. **Загрузчик (loader):** программа, обычно входящая в состав операционной системы, предназначенная для запуска процесса операционной системы на основе загрузочного модуля.
   Задача загрузчика – преобразовать условные адреса разделов памяти в абсолютные, используя специальную таблицу, которую редактор связей вставляет в заголовок исполняемого файла. Формат таблицы трансляции адресов зависит не только от архитектуры вычислительной системы, но и от той операционной системы, которая должна управлять выполнением готовых программ.

10. **Общая схема преобразования исходного кода в процесс операционной системы:**



11. **Алфавит языка:** базовый набор символов, разрешенных к использованию языком, который основывается на одной из кодировок.

ASCII: American Standard Code for Information Interchange — американский стандартный код для обмена информацией. ASCII представляет собой 8-битную кодировку для представления десятичных цифр, символов латинского и национального алфавитов, знаков препинания и управляющих символов. Нижнюю половину кодовой таблицы (0 - 127) занимают символы US-ASCII, а верхнюю (128 - 255) — другие нужные символы (CP866, CP1251).

Windows-1251: русская Windows-кодировка (CP1251, Windows-1251). Windows-1251 — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.

UNICODE: Unicode Consortium, 1991, ISO/IEC 10646, последняя версия 7.0. Юникод – стандарт кодирования символов, состоящий из 2х разделов:

- UCS - universal character set (универсальный набор символов задаёт однозначное соответствие символов кодам);
- UTF - Unicode transformation format (семейство кодировок - определяет машинное представление последовательности кодов UCS).

Кодировки: **UTF-8**, **UTF-16 (LE/BE)**, **UTF-32**.

BOM (маркер последовательности байтов) — юникод-символ, используемый для указания порядка следования байтов текстового файла.

12. **Термины и основные понятия языков программирования:**

Идентификатор: имя компонента программы (переменной, функции, метки, типа и пр.), составленное программистом по определенным правилам. В разных языках программирования правила составления идентификаторов обычно различаются. Например, в Microsoft Transact-SQL все имена переменных должны начинаться с символа **@**.

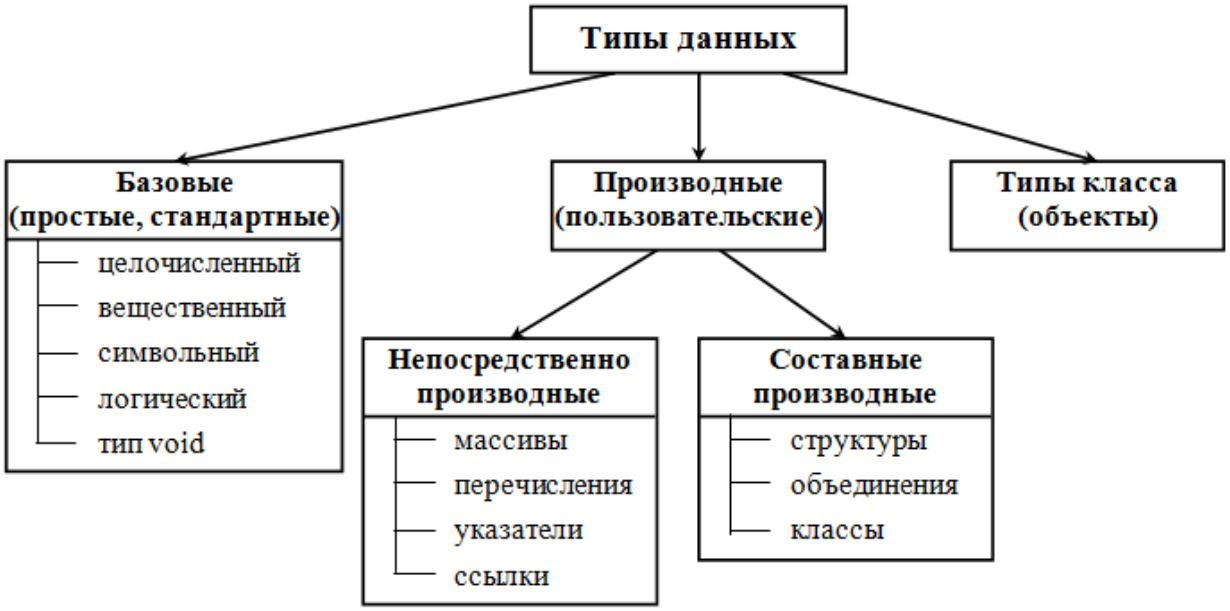
Зарезервированные идентификаторы: идентификаторы, которые предварительно определены в системе программирования.

Ключевые слова: последовательности символов алфавита языка, имеющие специальное назначение, зарезервированные для обозначения типов переменных, класса хранения, элементов операторов.

Типы данных: фундаментальные (базовые, встроенные), определенные программистом (пользовательские типы данных).

Тип данных определяет представление данных, хранение их в памяти и операции, которые можно с ними выполнять.

Литералы - неизменяемые величины (логические, целые, вещественные, символьные, строковые константы, дата и время). Компилятор, выделив константу в качестве лексемы, относит ее к одному из типов данных по ее внешнему виду. Программист может задать тип константы и явным образом.



Для простых типов данных определяются границы диапазона и количество байт, занимаемых ими в памяти компьютера.

Описание типа	Размер (биты)	Диапазон	.NET	C#	VB.NET	C/C++	Java
Байт со знаком	8	от -128 до 127	SByte	sbyte	SByte	signed char	byte
Байт без знака	8	от 0 до 255	Byte	byte	Byte	unsigned char	-
Короткое целое число со знаком	16	от -32 768 до 32 767	Int16	short	Short	short	short
Короткое целое число без знака	16	от 0 до 65 535	UInt16	ushort	UShort	unsigned short	-
Среднее целое число со знаком	32	от -2147483648 до 2147483647	Int32	int	Integer	int	int
Среднее целое число без знака	32	от 0 до 4294967295	UInt32	uint	UInteger	unsigned int	-
Длинное целое число со знаком	64	от -9223372036854775808 до 9223372036854775807	Int64	long	Long	long	long
Длинное целое число без знака	64	от 0 до 18446744073709551615	UInt64	ulong	Ulong	unsigned long	-
Вещественное число одинарной точности с плавающей запятой	32	$\pm 1{,}5*10^{-45}$ до $\pm 3{,}4*10^{33}$	Single	float	Single	float	float
Вещественное число двойной точности с плавающей запятой	64	$\pm 5*10^{-324}$ до $\pm 1{,}7*10^{306}$	Double	double	Double	double	double
Символ (8 бит)	8	ASCII	-	-	-	char	-
Символ (16 бит)	16	UNICODE	Char	char	Char	wchar_t	char
Логический тип	8	{true, false}	Boolean	bool	Boolean	bool	boolean
Строка	-	-	String	string	String	char*/wchar_t*	String
Пустой тип	-	-	-	void	-	void	void

Массивы данных фундаментальных типов: коллекция однородных данных, размещенных последовательно в памяти, доступ к которым осуществляется по индексу.

Пользовательские типы: типы создаваемые пользователем, всегда должно быть объявление типа.

Инициализация переменных (памяти): присвоение значения в момент объявления переменной; как правило, применяется литералы. Отличие от присвоения: при присвоении явно перемещаются данные. Инициализация массивов, структур. Функциональный вид инициализации.

Область видимости переменных: доступность переменных по их идентификатору в разных частях (блоках программы).

Преобразование типов: автоматическое (неявное) преобразование, явное преобразование. Неявное преобразование выполняется транслятором (компилятором или интерпретатором) по правилами, описанным в стандарте языка.

Константное выражение: выражение, которое должно быть вычислено на этапе компиляции.

Выражение: объединение литералов, имен (переменных, функций и пр.), операторов и специальных символов, служащих для вычисления выражения или достижения побочных эффектов (например: при применении в выражении функций).

lvalue – именующее выражение – это ссылка на значение – может использоваться в левой и правой части оператора присваивания (имя переменной, ссылка на элемент массива по индексу, вызов функции возвращающей указатель, всегда связано с областью памяти, адрес которой известен).

rvalue – значащее выражение – может использоваться только в правой части оператора присваивания (не связано с адресом, связано только со значением, например: литералы, вызов функции, возвращающей значение).

Пространство имен: именованная область видимости. Применяется для разрешения конфликтов имен.

Классы памяти: код, стек, статические данные, динамическая область. Принцип разделения памяти.

Система обработки исключений: исключение – событие при выполнении программы, при котором ее дальнейшее выполнение становится бессмысленным.

Оператор языка: законченное описание некоторого действия.
Операторы делят на исполняемые и неисполняемые, простые и составные. Исполняемые операторы задают действия над данными. Неисполняемые операторы служат для описания данных.
Составной оператор или блок - это группа операторов, заключенная в операторные скобки. Блоки могут быть вложенными.

Инструкции языка: инструкции объявления (if (int) условное объявление), составные инструкции ({}), инструкции выбора (if, switch), циклы (while, do while, for), инструкции переходов (goto, break, continue, return), инструкции обработки исключений (try, catch, throw).

Препроцессор: часть транслятора, которая выполняется до процесса трансляции, выполнят директивы препроцессора. Результатом выполнения препроцессора является текст, сформированный из исходного под действием директив препроцессора.

Программные конструкции: программные блоки**,** процедуры, функции, механизмы передачи параметров, возврат значений, области видимости, перегружаемые функции.

Программные конструкции: соглашения о вызове: способ передачи параметров, порядок размещения параметров, очищение стека, порядок возврата значения и пр.

Статическая библиотека: файл (обычно с расширением **lib**), содержащий объектные модули. Является входным файлом для компоновщика (**linker**).

Стандартная библиотека: как правило, в составе языка программирования есть обязательный (стандартный) набор функций. Такие функции называют встроенными функциями. Встраиваться функции могут тремя способами:

1) встраиваться прямо в код транслятора;
2) в отдельной библиотеке;
3) сочетание первого и второго случаев.

13. **Теория формальных языков. Основные понятия.**

Определения 1.

- ✓ I - алфавит, конечное множество символов, например $I = \{a, b, c\}$
- ✓ Цепочка α в алфавите I - конечная последовательность символов из алфавита I, например: $\alpha = abc$, $\beta = aaaa$, $\gamma = cbbb$.
- ✓ Пустая цепочка λ - цепочка, не содержащая ни одного символа.
- ✓ Длина цепочки $|\alpha|$. $|\lambda| = 0$.
- ✓ Равенство цепочек $\alpha = \beta$, если они имеют один и тот же состав символов, одно и то же количество символов $|\alpha| = |\beta|$ и тот же порядок символов.
- ✓ Конкатенация цепочек α и β - цепочка $\alpha\beta$, образованная их соединением, например $\alpha = abc$, $\beta = aaaa$, $\alpha\beta = abcaaaa$.
- ✓ Если $\alpha\beta\gamma$ - конкатенация цепочек α, β и γ, то α, β, γ - подцепочки $\alpha\beta\gamma$, α - префикс цепочки $\alpha\beta\gamma$, $\alpha\beta\gamma$ - суффикс цепочки $\alpha\beta\gamma$.
- ✓ Итерация цепочки α - α^n, $\alpha^0 = \lambda$.
- ✓ Положительное замыкание: если I - алфавит, то I^+ - множество всех цепочек, состоящих из символов I. $\lambda \notin I^+$.
- ✓ Замыкаеие Клини: если I - алфавит, то I^* - множество всех цепочек, состоящих из символов I. $\lambda \in I^*$. $I^* = I^+ \cup \lambda$.

Определения 2.

- ✓ Язык $L(I)$ над алфавитом I - произвольное множество цепочек из I^*. $L(I) \subseteq I^*$.
- ✓ Язык $L_1(I)$ является подмножеством языка $L_2(I)$, если каждая цепочка в язык L_1 входит в язык L_2, язык L_2 включает язык L_1. $L_1(I) \subseteq L_2(I) \Leftrightarrow (\forall \alpha \in L_1(I) \Rightarrow \alpha \in L_2(I))$
- ✓ Эквивалентность языков: языки $L_1(I)$ и $L_2(I)$ совпадают, если язык $L_1(I)$ включает язык $L_2(I)$ и язык $L_2(I)$ включает язык $L_1(I)$. $L_1(I) = L_2(I) \Leftrightarrow (L_1(I) \subseteq L_2(I) \wedge L_2(I) \subseteq L_1(I))$.

Способы задания формальных языков: язык L можно определить тремя способами:

- перечислением всех цепочек языка;
- указанием способа (алгоритма) порождения цепочек;
- определить метод (алгоритм) распознавания цепочек.

Лексика языка программирования – множество цепочек языка.
Синтаксис языка – набор формальных правил, определяющий конструкции (последовательности цепочек).
Семантика языка - набор неформальных правил (невозможно записать правила в виде формальных выражений), которые описываются словесно (например, в руководстве программиста). Пример: применению переменной должно предшествовать ее объявление, при конвертации типов следует обеспечить соответствующее значение переменной.

Чтобы **создать** язык программирования, следует определить:
- множество допустимых символов (алфавит);
- формально описать множество правильных программ;
- задать семантические правила языка.

14. **Порождающая грамматика** – это четверка
$G = \langle T, N, P, S \rangle,$
где
T - множество терминальных символов,
N - множество нетерминальных символов,
P - множество правил (говорят продукций) грамматики,
S - начальный символ грамматики.

Определения 3.

- ✓ Цепочка $\beta \in (N \cup T)^*$ непосредственно выводима из цепочки $\alpha \in (N \cup T)^+$ в грамматике $G = \langle T, N, P, S \rangle$, если $\alpha = \mu\gamma\tau$, $\beta = \mu\delta\tau$, где $\delta \in (N \cup T)^*$, $\gamma \in (N \cup T)^+$ и правило вывода $\gamma \to \delta$ содержится в P. Записывается $\alpha \Rightarrow \beta$.
- ✓ Запись $\alpha \Rightarrow^* \beta$ предполагает $n \geq 0$ шагов вывода β из α. В том случае, если $\alpha \Rightarrow \beta$, то число шагов вывода $n = 0$.
- ✓ Запись $\alpha \Rightarrow^+ \beta$ предполагает $n > 0$ шагов вывода β из α.
- ✓ Если $S \Rightarrow^* \beta$ и $\beta \in (T \cup N)^*$,то β называется **сентенциальной** формой грамматики $G = \langle T, N, P, S \rangle$.
- ✓ Если $S \Rightarrow^* \beta$ и $\beta \in T^*$,то β называется терминальной сентенциальной формой грамматики $G = \langle T, N, P, S \rangle$.
- ✓ $L(G)$ - язык, порождаемый грамматикой G. Язык $L(G)$содержит все терминальные цепочки, выводимые из S: $L(G) = \{\alpha \in T^* \mid S \Rightarrow^* \alpha\}$.

$L(G)$ - множество терминальных сентенциальных форм грамматики G.

- ✓ $G_2 = G_1 \Leftrightarrow L(G_2) = L(G_1)$ - грамматики эквивалентны, если они порождают один язык.

Способы задания грамматик: аналитическая форма, форма Бэкуса-Наура, синтаксическая диаграмма.

15. **Иерархия Хомского**: $G_0 \supset G_I \supset G_{II} \supset G_{III}$,
где $G_0, G_I, G_{II}, G_{III}$ - множества грамматик типа 0, 1, 2 и 3.

- ✓ Грамматики типа 0: $G_0 = \langle T, N, P, S \rangle$ - неограниченные грамматики, у которых нет никаких ограничений для правил. Правила имеют вид: $\alpha \to \beta$, где $\alpha \in V^+$, $\alpha \in V^*$.
- ✓ Грамматики типа 1: $G_I = \langle T, N, P, S \rangle$ - контекстно-зависимые грамматики (неукорачивающие грамматики). Правила имеют вид: $\alpha \to \beta$, где $\alpha \in V^+$, $\alpha \in V^*$ и $|\alpha| \le |\beta|$. Контекстно-зависимая, т.к. один и тот же нетерминальный символ может быть заменен на ту или иную цепочку символов в зависимости от контекста (цепочки) в которой он встречаются.
- ✓ Грамматики типа 2: $G_{II} = \langle T, N, P, S \rangle$ - контекстно-свободные грамматики. Правила имеют вид: $A \to \alpha$, где $A \in N$, $\alpha \in V^*$.
- ✓ Грамматики **типа 3**: $G_{III} = \langle T, N, P, S \rangle$ - **регулярные грамматики.** Регулярные грамматики бывают праволинейными и леволинейными. Правила *праволинейной* грамматики имеют вид: $A \to \alpha$ или $A \to \alpha B$, где $A, B \in N$, $\alpha \in T^*$.
  Правила *леволинейной* грамматики имеют вид: $A \to \alpha$ или $A \to B\alpha$, где $A, B \in N$, $\alpha \in T^*$.

16. Соотношения грамматик в иерархии Хомского:

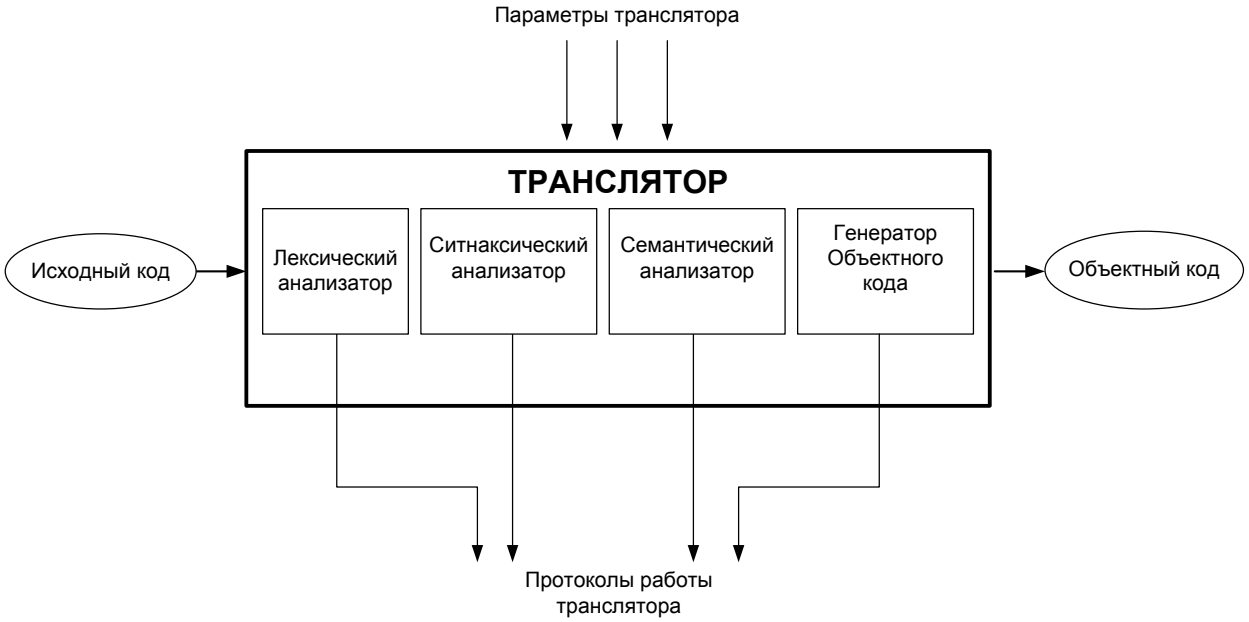
1) любая регулярная грамматика является контекстно-свободной грамматикой;
2) любая контекстно-свободная грамматика является контекстно-зависимой грамматикой;
3) любая контекстно-зависимая грамматика является грамматикой типа 0.

Формальные языки классифицируются по типу порождающих их грамматик.

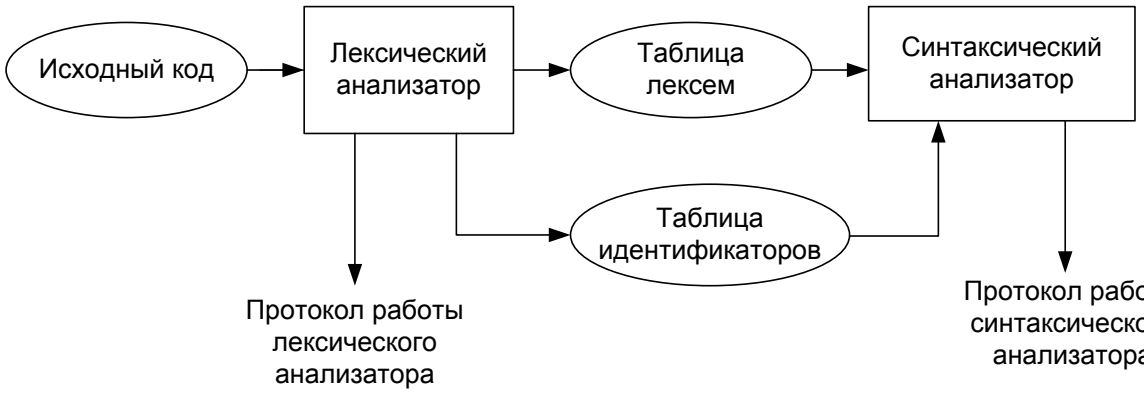
Между типами формальных языков существуют следующие соотношения:

1) каждый регулярный язык является контекстно-свободным языком, но существуют контекстно-свободные языки, которые не являются регулярными;
2) каждый контекстно-свободный язык является контекстно-зависимым, но существуют контекстно-зависимые, которые не являются контекстно-свободными.
3) каждый контекстно-зависимый язык является языком типа 0.

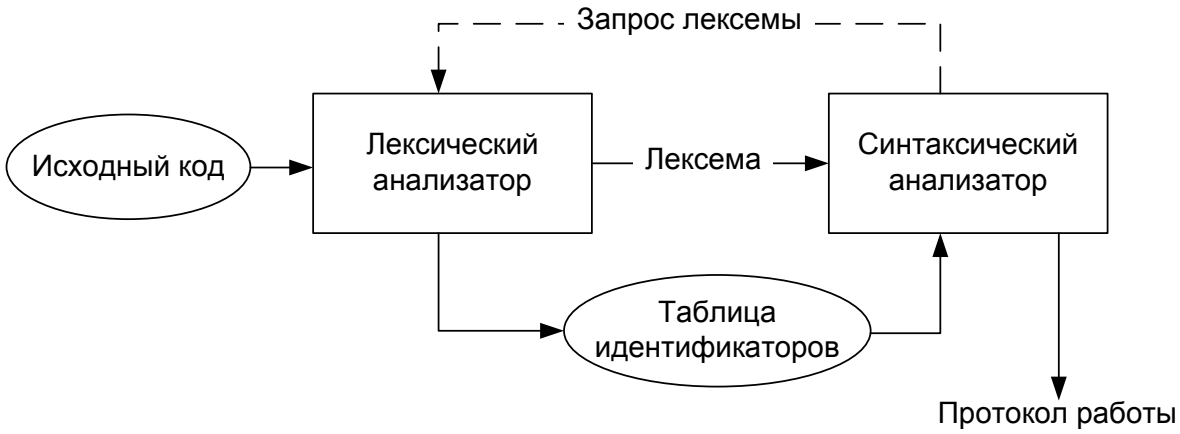
17. **Структура транслятора**:



18. **Лексический анализ** – первая (наиболее простая) фаза трансляции. Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором (сканером).
Взаимодействие лексического и синтаксического анализаторов: последовательное или параллельное.
Последовательное взаимодействие лексического и синтаксического анализаторов:



Параллельное взаимодействие лексического и синтаксического анализаторов:



Грамматика описывает множество правильных цепочек символов над заданным алфавитом.

Регулярное выражение описывает множество цепочек – формальный язык. Для записи регулярного выражения используются метасимволы. Множество цепочек описанных регулярным выражением называется **регулярным множеством** (или регулярным языком).

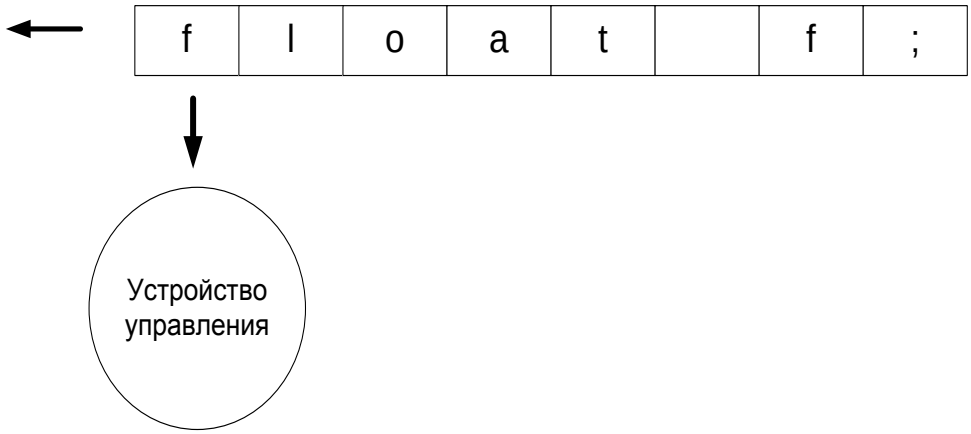
Определение регулярного множества

Пусть I – алфавит.
Регулярные выражения над алфавитом I и языки, представляемые ими, рекурсивно определяются следующим образом:

1) $\varnothing$ – регулярное выражение и представляет пустое множество;
2) λ – регулярное выражение и представляет множество $\{\lambda\}$;
3) для каждого $a \in I$ символ a является регулярным выражением и представляет множество $\{a\}$;
4) если p – регулярное приложение, представляющее множество P, если q – регулярное приложение, представляющее множество Q, то $p+q$, pq, q^* являются регулярными выражениями и представляют множества $P \cup Q$, PQ (конкатенация множеств)и P^* соответственно.
5) $pp^* = p^+$

Символы, применяемые для описания регулярных выражений, называются **метасимволами** или **символами-джокерами**. В описанном выше языке джокерами являются символы: $*,^{+},+,(,),\varnothing$.

Схема работы лексического анализатора



19. Класс алгоритмов, соответствующих схеме (см. выше), могут быть записаны в форме конечного автомата (КА).

✓ КА является дискретной системой, работающей по тактам (шагам).
✓ На вход КА подается входная цепочка, состоящая из символов входного алфавита.
✓ На каждом шаге КА находится в одном из возможных состояний, которое называется текущим.
✓ Шаг работы КА состоит в переходе из текущего состояния в новое состояние при получении на вход очередного символа входной цепочки. Этот переход определяется функцией переходов.
✓ Результат работы КА заключается в формировании выходного символа, который определяется парой «текущее состояние – входной символ». Выходной символ может быть пустым.

Определение КА.

КА это пятерка $M = (S, I, \delta, s_0, F)$,
где
S – конечное множество состояний устройства управления;
I – алфавит входных символов;
δ – функция переходов, отображающая $S \times (I \cup \{\lambda\})$ в множество подмножеств S: $\delta(s,i) \subset S, s \in S, i \in I$;
$s_0 \in S$ – начальное состояние устройства управления;
$F \subseteq S$.– множество заключительных (допускающих) состояний устройства управления.

Если $\delta(s,\lambda)=\varnothing$ и $|\delta(s,a)|\leq 1$, то конечный автомат **детерминированный** (ДКА) иначе – конечный автомат **недетерминированный** (НКА).

Определения.

- ✓ Мгновенное описание КА является пара (s,w), где $s\in S$ – состояние КА, $w\in I^*$ – неиспользованная часть входной цепочки.
- ✓ (s_0,w_0)- начальное мгновенное описание КА, w_0 – анализируемая цепочка.
- ✓ $(s_f,\lambda), s_f\in S$ - допускающее мгновенное описание КА.
- ✓ Если (s,aw) и $s'\in\delta(s,a)$, где $s',s\in S$, $a\in I\cup\lambda$, $w\in I^*$, то $(s,aw)\succ(s',w)$ - непосредственно следует.
- ✓ Если $(s_i,w_i)\succ(s_{i+1},w_{i+1})\succ(s_{i+2},w_{i+2})\succ\ldots\succ(s_k,w_k)$, то $(s_i,w_i)\succ *(s_k,w_k)$ - следует.
- ✓ Если $(s_0,w)\succ *(s_f,\lambda)$, $s_0\in S$ - начальное состояние, $s_f\in F$ - конечное состояние, то цепочка $w\in I^*$ допускается (распознается) КА.

Конечный автомат может быть однозначно задан своим графом переходов.
Доказаны 4 утверждения:

1) язык является регулярным множеством тогда и только тогда, когда он задан регулярной грамматикой;
2) язык может быть задан регулярной грамматикой (левосторонней или правосторонней) тогда и только тогда, когда язык является регулярным множеством;
3) язык является регулярным множеством тогда и только тогда, когда он задан конечным автоматом;
4) язык распознается с помощью конечного автомата тогда и только тогда, когда он является регулярным множеством.

Другими словами: любой регулярный язык может быть задан регулярной грамматикой, регулярным выражением или конечным автоматом.
Другими словами: любой конечный автомат задает регулярный язык, а значит грамматику или регулярное выражение.

Доказана **теорема** (А. Ахо, Дж. Хопкрофт, Дж. Ульман):

пусть α - регулярное выражение, тогда найдется недетерминированный конечный автомат $M = (S, I, \delta, s_0, \{s_f\})$, допускающий автомат, представленный α, и обладающий следующими свойствами:

1) $|S| \le 2|\alpha|$;

2) $\forall a \in I \cup \{\lambda\} : \delta(s_f, a) = \varnothing$;

3) $\forall s \in S : \sum_{a \in I \cup \{\lambda\}} |\delta(s, a)| \le 2$.

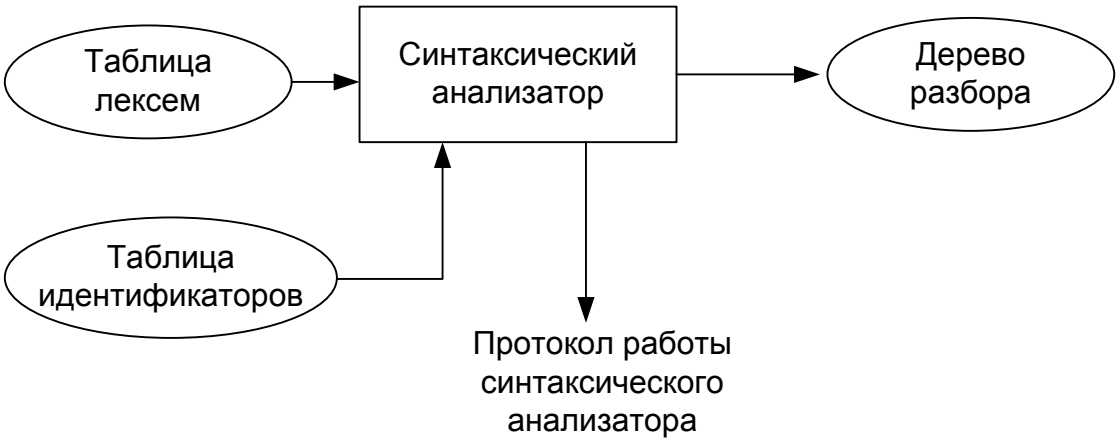
- ✓ **Построение графа конечного автомата по регулярному выражению.**
- ✓ **Алгоритм для разбора с двумя массивами.**

20. **Синтаксический анализ:** фаза трансляции, выполняемая после фазы лексического анализа и предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода.

    Синтаксический анализ: основная фаза трансляции, без нее процесс трансляции не имеет смысл. Все задачи лексического анализа могут быть решены в рамках синтаксического анализа. Т.е. можно создать транслятор без лексического анализатора. Лексический анализ необходим для освобождения алгоритма синтаксического разбора от рутинных алгоритмов.

    Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ. Входом для синтаксического анализа является таблица лексем (токенов) и таблица идентификаторов. Выходом – дерево разбора.

    Входная и выходная информация синтаксического анализатора.



    Задачи, выполняемые синтаксическим анализатором:

1) поиск и выделение синтаксических конструкций в исходном тексте (разбор);
2) распознавание (проверка правильности) синтаксических конструкций;
3) выявление ошибок и продолжение процесса распознавания после обработки ошибок;
4) если нет ошибок, формирование дерева разбора.

Исходный текст программы для синтаксического анализатора – таблица лексем (токенов).

Для описания языка, разбираемого синтаксическим анализатором, применяют грамматики типа 2 – контекстно-свободные грамматики (лекция 12).

Правила контекстно-свободных грамматик.

Грамматики типа 2: $G_{II} = \langle T, N, P, S \rangle$ – контекстно-свободные грамматики, правила которых имеют вид: $A \to \alpha$, где $A \in N$, $\alpha \in V^*$, $V = N \cup T$ - словарь грамматики G_{II}.

Три типа синтаксических анализаторов:

- нисходящие;
- восходящие;
- универсальные.

Приведение грамматик

Грамматика G является приведенной грамматикой, если в грамматике нет:

- бесплодных символов;
- недостижимых символов;
- λ-правил;
- цепных правил.

Приведение грамматики G - отыскание эквивалентной приведенной грамматики G'. Процесс приведения - это упрощение грамматики.

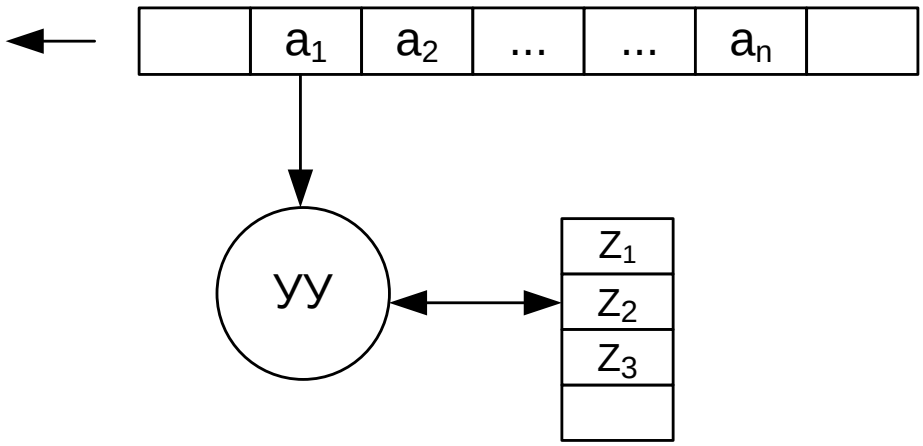
Алгоритм удаления бесплодных символов.

Алгоритм удаления недостижимых символов (перед удалением недостижимых символов, должны быть удалены бесплодные символы).

Алгоритм удаления λ-правил.

Алгоритм исключения цепных правил.

21. Автомат с магазинной памятью (МП-автоматы): распознаватели контекстно-свободных языков.



Формальное описание МП-автомата:

$M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$

Q - множество состояний;

V - алфавит входных символов;

Z - специальный алфавит магазинных символов;

δ-функция переходов автомата $Q \times (V \cup \{\lambda\}) \times Z \to P(Q \times Z^*)$, где $P(Q \times Z^*)$ - множество подмножеств $Q \times Z^*$;

$q_0 \in Q$ - начальное состояние автомата;

$z_0 \in Z$ - начальное состояние магазина (маркер дна);

$F \subseteq Q$ - множество конечных состояний.

Определения

- ✓ Конфигурация (текущее состояние автомата) описывается тройкой (q, α, ω), где q - текущее состояние автомата, α - остаток цепочки, ω - цепочка-содержимое магазина.
- ✓ Один такт работы автомата $(q, a\alpha, z\omega) \succ (q', \alpha, \gamma\omega)$, если $(q', \gamma) \in \delta(q, a, z)$.
- ✓ Начальное состояние (q_0, α, z_0), q_0 - начальное состояние автомата, α - входная цепочка, z_0 - маркер дна магазина.
- ✓ Цепочка α является допустимой (распознается) автоматом $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$, если $(q_0, \alpha, z_0) \succ^* (q', \lambda, \lambda)$ и $q' \in F$.

29. **Работа автомата** $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$

1) состояние автомата $(q, a\alpha, z\beta)$

2) читает символ a находящийся под головкой (сдвигает ленту);
3) не читает ничего (читает λ, не сдвигает ленту);
4) из δ определяет новое состояние q', если $(q',\gamma)\in\delta(q,a,z)$ или $(q',\gamma)\in\delta(q,\lambda,z)$.
5) читает верхний (в стеке) символ z и записывает цепочку γ т.к. $(q',\gamma)\in\delta(q,a,z)$, при этом, если $\gamma=\lambda$, то верхний символ магазина просто удаляется.
6) работа автомата заканчивается (q,λ,λ)

На каждом шаге автомата возможны три случая:

1) функция $\delta(q,a,z)$ определена – осуществляется переход в новое состояние;
2) функция $\delta(q,a,z)$ не определена, но определена $\delta(q,\lambda,z)$ – осуществляется переход в новое состояние (лента не продвигается);
3) функции $\delta(q,a,z)$ и $\delta(q,\lambda,z)$ не определены – дальнейшая работа автомата не возможна (цепочка не разобрана).

Язык $L(M)=\{\alpha\mid(q_0,\alpha,z_0)\succ^{*}(q',\lambda,\lambda),q'\in F\}$ – допускаемый автоматом M.

Для построения МП-автомата необходимо привести контекстно-свободную грамматику к одной из нормальных форм: к **нормальной форме Хомского** или **нормальной форме Грейбах**.

30. **Нормальная форма Хомского.**

Контекстно-свободная грамматика $G=\langle T,N,P,S\rangle$ имеет нормальную форму Хомского, если правила P имеют вид:

1) $A\to BC$, где $A,B,C\in N$;
2) $A\to a$, где $A\in N, a\in T$;
3) $S\to\lambda$, где $S\in N$ - начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

Алгоритм преобразования контекстно-свободной грамматики $G=\langle T,N,P,S\rangle$ к грамматике $G'=\langle T,N',P',S\rangle$ в нормальной форме Хомского.

Праворекурсивное правило: правило вида $A\to\alpha A$,

где $\alpha\in(T\cup N)^{*}$, $A\in N$

Леворекурсивное правило: правило вида $A\to A\alpha$,

где $\alpha\in(T\cup N)^{*}$, $A\in N$

Для каждой грамматики $G = \langle T, N, P, S \rangle$, содержащей леворекурсивные правила можно построить грамматику $G' = \langle T, N', P', S \rangle$ не содержащую леворекурсивное правило.

Для каждой грамматики $G = \langle T, N, P, S \rangle$, содержащей правокурсивные правила можно построить грамматику $G' = \langle T, N', P', S \rangle$ не содержащую правокурсивное правило.

31. **Нормальная форма Грейбах:**

Контекстно-свободная грамматика $G = \langle T, N, P, S \rangle$ имеет нормальную форму Грейбах, если она не является леворекурсивной (не содержит леворекурсивных правил) и правила P имеют вид:

1) $A \to a\alpha$, где $a \in T, \alpha \in N^*$;
2) $S \to \lambda$, где $S \in N$ - начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

Алгоритм устранения левой рекурсии.

Пусть правило грамматики $G = \langle T, N, P, S \rangle$ имеет вид:

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

Введем новый нетерминал B и преобразуем леворекурсивную продукцию в эквивалентные правила без левой рекурсии

$A \to \beta B$

$B \to \alpha B \mid \varepsilon$

Оставим без изменения строки, порождаемые из A

Пример устранения левой рекурсии.

Пусть правила P грамматики G имеют вид:

$A \to A + A \mid x$

$x, x + x, x + x + x, x + x + x + x, \ldots$

Преобразование:

$A \to x \mid xA'$

$A' \to +A \mid +AA'$

$x, x + x, x + x + x, x + x + x + x, \ldots$