**Лекция 13**

**БГТУ, ФИТ, ПОИТ, 2 семестр, Языки программирования**

**Лексический анализ**

1. **Лексический анализ** – первая (наиболее простая) фаза трансляции. Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором (сканером).
2. Структура транслятора:



1. Взаимодействие лексического и синтаксического анализаторов: последовательное или параллельное.
2. Последовательное взаимодействие лексического и синтаксического анализаторов.



1. Параллельное взаимодействие лексического и синтаксического анализаторов.



1. ***Пример:***



1. Взаимодействие лексического и синтаксического анализаторов С, С++, Java последовательное.
2. ***Пример:*** грамматика  для **float id**

,,























- регулярная праволинейная грамматика



1. ***Пример:*** вывод (распознавание) цепочки **float f;**

|  |  |
| --- | --- |
| Вывод цепочки | Дерево нисходящего разбора цепочки |
|  |  |

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

1. ***Пример:*** вывод (распознавание) цепочки **float f2x =**

|  |  |
| --- | --- |
| Вывод цепочки | Дерево нисходящего разбора цепочки |
|  |  |

1. ***Пример:*** вывод (распознавание) цепочки **float 22x =**





**! нет правила вывода – цепочка не распозналась**

1. ***Пример:*** грамматика  для **float f;**

,,















1. - регулярная леволинейная грамматика.



Вывод называется *левосторонним,* если в нем на каждом шаге вывода правило грамматики применяется всегда к крайнему левому нетерминальному символу в цепочке. Другими словами – на каждом шаге вывода происходит подстановка цепочки символов на основании правила грамматики, т.е. вместо крайнего левого нетерминального символа в исходной цепочке.

1. ***Пример:*** восходящий вывод (распознавание) цепочки **float f;**

|  |  |
| --- | --- |
| Вывод цепочки | Дерево восходящего разбора цепочки |
|  |  |

1. Пример: вывод (распознавание) цепочки **float f2x =**













Восходящий вывод цепочки

1. Для описания лексики языка программирования, обычно применяются регулярные грамматики. Другими словами, с точки зрения лексического анализатора – язык программирования набор лексем (токенов), которые распознаются (классифицируются) лексическим анализатором. Или, иначе говоря, язык программирования (на уровне лексического анализа) представляет собой регулярный язык (язык типа 3 иерархии Хомского).
2. Грамматика языка описывает множество правильных цепочек символов над заданным алфавитом.

Как правило, для описания регулярных языков не применяют грамматики в виду громоздкости записи, а используют другую форму – регулярные выражения.

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

|  |
| --- |
| Пусть  – алфавит.  Регулярные выражения над алфавитом  и языки, представляемые ими, рекурсивно определяются следующим образом:   1. – регулярное выражение, представляет пустое множество; 2. – регулярное выражение, представляет множество ; 3. для каждого  символ  является регулярным выражением и представляет множество ; 4. если  – регулярное выражение, представляющее множество  и   – регулярное выражение, представляющее множество , то  , ,  являются регулярными выражениями и представляют множества:    1. (объединение),    2. (конкатенация множеств),    3. (итерация) соответственно. |

1. Символы, применяемые для описания регулярных выражений, называются **метасимволами** или **символами-джокерами**. В описанном выше языке джокерами являются символы: .
2. ***Примеры***

|  |  |
| --- | --- |
| Регулярное выражение | Множество |
| a | a |
| a+b | a,b |
| a+b+c | a,b,c |
| a+ | a, aa, aaa, aaaa... |
| a\* | λ, a, aa, aaa, aaaa,... |
| ab | ab |
| ab+cd | ab, cd |
| (ab+cd)+ | ab, cd, abab, abcd, cdcd, cdab, ababab, cdcdcd, abcdab, cdabcd, abcdcdcdcd, abababab,... |
| (ab+cd)\* | λ, (ab+cd)+ |
| a(bc+de) | abc, ade |
| a(bc+de)f | abcf, adef |
| ab+c | abc, abbc, abbbc, abbbbc,… |
| ab\*c | ac, abc, abbc, abbbc, abbbbc,… |
| a(bc+de)+f | abcf, adef, abcbcf, abcdef, adebcf, adedef, abcdedcdef,... |
| a(bc+de)\*f | af, a(bc+de)+f |
| (ab+cd)(ef+gh) | abef,abgh, cdef, cdgh |
| (ab+cd)e+ | abe, cde,abee, cdee, abeee, cdeee,... |
| (ab+cd)e\* | ab, cd, (ab+cd)e+ |

1. ***Пример:*** **float f**



1. Теория регулярных языков была разработана в 1940х годах.

Нейрофизиологи **Уоррен Мак-Каллох** и **Уолтер Питс** моделировали нервную систему на нейронном уровне. Математик **Стивен Клин** формально описал модели нейрофизиологов с помощью алгебры, которую назвал регулярными множествами. Для формального описания этих множеств он разработал простую математическую запись, которую назвал регулярным языком. Запись использует специальные символы, которые в настоящее время называют метасимволами или символами-джокерами.

1. На сегодняшний день существует несколько диалектов регулярных языков (наборов метасимволов):

**grep** (global regular expression print) – команда в uniх/linux;   
**egrep** (extended grep) – разработал Aльфред Ахо;

**BRE** (Basic Regular Expression) и **ERE** (Extended regular expression) – BRE + POSIX (Portable Operating System Interface) – попытка стандартизировать; **Perl** – встроенный в язык Perl регулярные выражения;

**ECMA–стандарт** регулярных выражений в JavaScript;

**SED** (Stream Editor) – Bell Labs (1973-74 Lee E. McMahon).

1. В настоящее время чаще всего используется Perl-нотация (набор метасимволов) регулярных выражений. В том числе в стандартных библиотеках С++ и C#.
2. В стандартной библиотеке С++ набор функций **<regex>**.
3. Пример применения функции **regex\_math**:



1. Схема работы лексического анализатора



1. Класс алгоритмов, соответствующих приведенной выше схеме, может быть записан в форме конечного автомата (KA).
2. ***Определение.***

КА это пятерка ,где

 – конечное множество состояний устройства управления;

 – алфавит входных символов;

 – функция переходов, отображающая  во множество подмножеств : ;

- начальное состояние устройства управления;

- множество заключительных (допускающих) состояний устройства управления.

Если  и , то конечный автомат – **детерминированный** (ДКА).

Т.е. отсутствуют состояния, имеющие λ-переходы и для каждого состояния *s* и входного символа *а* существует не более одной дуги, выходящей из *s* и помеченной как *а*. ДКА – это автомат, который переходит из любого состояния по любому символу точно в одно состояние.

Иначе ‑ конечный автомат является **недетерминированным** (НКА).

1. Мгновенное описание КА – пара ,

где  – состояние КА, – неиспользованная часть входной цепочки.

1.  – начальное мгновенное описание КА,  – анализируемая цепочка.
2.  – допускающее мгновенное описание КА.
3. Если  и , где , , , то – читается: непосредственно следует.
4. Если , то  – следует.
5. Если ,  – начальное состояние,  – конечное состояние, то цепочка  допускается (или распознается) КА.
6. Пример: пусть  входная цепочка,

КА , где функция  задана следующей таблицей.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Последовательность мгновенных описаний автомата



1.  – значит, что автомат  допускает (распознает) цепочку .
2. ***Определение.***

**Графом переходов** конечного автомата  называется ориентированный граф ,

где - множество вершин графа совпадает с множеством состояний конечного автомата,

– множество ребер (направленных линий, соединяющих вершины),

ребро , если .

Метка ребра  – все , для которых .



1. Конечный автомат может быть однозначно задан своим графом переходов.
2. 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. |

1. *Другими словами:* любой регулярный язык может быть задан регулярной грамматикой, регулярным выражением или конечным автоматом.
2. *Или:* любой конечный автомат задает регулярный язык, а значит грамматику или регулярное выражение.
3. Доказана теорема (А. Ахо, Дж. Хопкрофт, Дж. Ульман):

|  |
| --- |
| 1. пусть  - регулярное выражение, тогда найдется недетерминированный конечный автомат , допускающий автомат, представленный , и обладающий следующими свойствами: 2. 1) ; 3. 2) ; 4. 3) . |

1. Построение графа конечного автомата по регулярному выражению:

***Метод.*** Автомат для выражения строится композицией из автоматов, соответствующих подвыражениям. На каждом шаге построения строящийся автомат имеет в точности одно заключительное состояние, в начальное состояние нет переходов из других состояний и нет переходов из заключительного состояния в другие.









1. ***Пример:*** пусть язык  задан регулярным выражением 
2. Подвыражение: 



1. Продолжение. Подвыражение: 



1. Продолжение. Подвыражение: 



1. Выражение: 



1. Пример (продолжение): 



1. Пример (продолжение): 



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Пример (продолжение): алгоритм разбора

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |



1. Пример (продолжение): алгоритм разбора

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |



1. Пример (продолжение): алгоритм разбора

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |



1. Реализация алгоритма для разбора

