**Лабораторная работа №4**

**Способы записи синтаксиса языка**

**Цель работы**: изучить способы записи синтаксических правил.

**Бэкуса-Наура формы (БНФ)**

Метаязыки Хомского и Хомского-Щутценберже использовались в математической литературе при описании простых абстрактных языков. Метаязык, предложенный Бэкусом и Науром, впервые использовался для описания синтаксиса реального языка программирования Алгол 60. Наряду с новыми обозначениями метасимволов, в нем использовались содержательные обозначения нетерминалов. Это сделало описание языка нагляднее и позволило в дальнейшем широко использовать данную нотацию для описания реальных языков программирования. Были использованы следующие обозначения:

* символ "::=" отделяет левую часть правила от правой;
* нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки "<" и ">";
* терминалы - это символы, используемые в описываемом языке;
* каждое правило определяет порождение нескольких альтернативных цепочек, отделяемых друг от друга символом вертикальной черты "|".

Пример описания идентификатора с использованием БНФ:

1. <буква> :: = А|В|С|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|  
   W|X|Y|Z|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

1. <цифра> :: = 0|1|2|3|4|5|6|7|8|9
2. <идентификатор> ::= <буква> | <идентификатор><буква> |  
   <идентификатор><цифра>

Правила можно задавать и раздельно:

1. <идентификатор> :: = <буква>
2. <идентификатор> :: = <идентификатор> <буква>
3. <идентификатор> :: = <идентификатор> <цифра>

**Расширенные Бэкуса-Наура формы (РБНФ)**

Метаязык, представленный выше, позволяет описывать любой синтаксис. Однако, для повышения удобства и компактности описания, целесообразно вести в язык дополнительные конструкции. В частности, специальные метасимволы были разработаны для описания необязательных цепочек, повторяющихся цепочек, обязательных альтернативных цепочек. Существуют различные расширенные формы метаязыков, незначительно отличающиеся друг от друга. Их разнообразие зачастую объясняется желанием разработчиков языков программирования по-своему описать создаваемый язык. К примерам таких широко известных метаязыков можно отнести: метаязык PL/I, метаязык Вирта, используемый при описании Модулы-2, метаязык Кернигана-Ритчи, описывающий Си. Зачастую такие языки называются расширенными формами Бэкуса-Наура (РБНФ).

В частности, РБНФ, используемые Виртом, имеют следующие особенности:

* Квадратные скобки "

**[**" и "**]**" означают, что заключенная в них синтаксическая конструкция может отсутствовать;

* фигурные скобки "

**{**" и "**}**" означают ее повторение (возможно, 0 раз);

* круглые скобки "

**(**" и "**)**" используются для ограничения альтернативных конструкций;

* сочетание фигурных скобок и косой черты "

**{/**" и "**/}**" используется для обозначения повторения один и более раз.

Нетерминальные символы изображаются словами, выражающими их интуитивный смысл и написанными на русском языке.

Если нетерминал состоит из нескольких смысловых слов, то они должны быть написаны слитно. В этом случае для повышения удобства в восприятии фразы целесообразно каждое ее слово начинать с заглавной буквы или разделять слова во фразах символом подчеркивания. Терминальные символы изображаются словами, написанными буквами латинского алфавита (зарезервированные слова) или цепочками знаков, заключенными в кавычки. Синтаксическим правилам предшествует знак "**$**" в начале строки. Каждое правило оканчивается знаком "**.**" (точка). Левая часть правила отделяется от правой знаком "**=**" (равно), а альтернативы - вертикальной чертой "**|**". Этот вариант РБНФ и будет использоваться для описания синтаксиса языков в лабораторной работе. В соответствии с данными правилами синтаксис идентификатора будет выглядеть следующим образом:

$ 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"|"a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"|"n"|"o"|

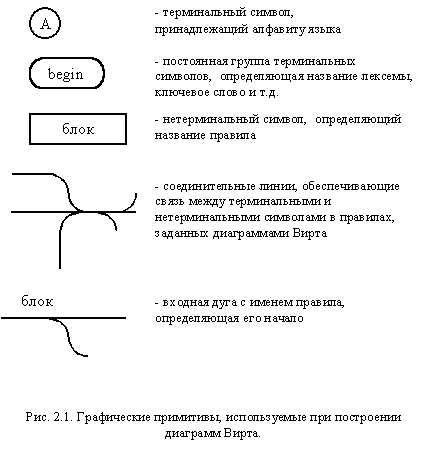
"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z".

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

$ идентификатор = буква {буква | цифра}.

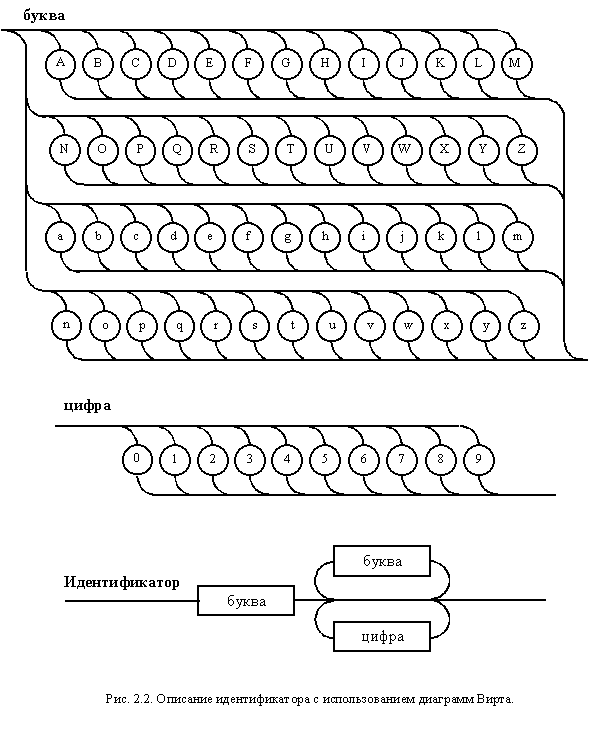
**Диаграммы Вирта**

Наряду с текстовыми способами описания синтаксиса языков широко используются и графические метаязыки, среди которых наиболее широкую известность получил язык диаграмм Вирта, впервые примененный для описания языка Паскаль. Метасимволы заменены следующими графическими обозначениями (см. рис. 2.1):



* терминальные символы и их постоянные группы располагаются в окружностях или прямоугольниках со скругленным вертикальными сторонами;
* нетерминальные символы заносятся внутрь прямоугольников;
* каждый графический элемент, соответствующий терминалу или нетерминалу, имеет по одному входу и выходу, которые обычно рисуются на противоположных сторонах;
* каждому правилу соответствует своя графическая диаграмма, на которой терминалы и нетерминалы соединяются посредством дуг;
* альтернативы в правилах задаются ветвлением дуг, а итерации - их слиянием;
* должна быть одна входная дуга (располагается обычно слева и сверху), задающая начало правила и помеченная именем определяемого нетерминала, и одна выходная, задающая его конец (обычно располагается справа и снизу).

Пример описания идентификатора с использованием диаграмм Вирта представлен на рисунке 2.2.



Обычно стрелки на дугах диаграмм не ставятся, а направления связей отслеживаются движением от начальной дуги в соответствии с плавными изгибами промежуточных дуг и ветвлений. Таким же образом определяются входы и выходы терминалов и нетерминалов. Специальных стандартов на диаграммы Вирта нет, поэтому графические обозначения могут меняться в зависимости от средств рисования. Можно, например, использовать псевдографику или просто текстовые символы, связи со стрелками. Однако такой вид правил менее удобен для восприятия и поэтому применяется крайне редко.

Диаграммы Вирта позволяют задавать альтернативы, рекурсии, итерации и по изобразительной мощности эквивалентны РБНФ. Но графическое отображение правил более наглядно. Кроме этого допускается произвольное проведение дуг, что уменьшает количество элементов в правиле за счет его неструктурированности. Диаграммы Вирта являются удобным исходным документом для построения лексического и синтаксического анализаторов.

**Порядок выполнения работы**

1. Для лексических конструкций из варианта задания разработать правила в виде РБНФ.
2. Разработанные правила преобразовать в диаграммы Вирта.
3. Оформить отчет, в котором привести результаты работы.

Вариант лабораторной работы соответствует варианту лабораторной работы №2.

**Варианты заданий**

1. Входной язык содержит арифметические выражения, разделенные символом ;(точка с запятой). Арифметические выражения состоят из идентификаторов, десятичных чисел с плавающей точкой (в обычной и логарифмической форме), знаков операций и скобок.
2. Входной язык содержит арифметические выражения, разделенные символом ;(точка с запятой). Арифметические выражения состоят из идентификаторов, римских чисел, знаков операций и скобок. (Римскими считать числа, записанные большими буквами X, V и I).
3. Входной язык содержит упрощенные операторы цикла типа  
   while <логическое выражение> do <оператор присваивания>; Логическое выражение может содержать идентификаторы, знаки операций сравнения, целые десятичные числа без знака, скобки и логические операции and и or. Оператор присваивания должен состоять из идентификатора, знака присваивания и целой десятичной константы без знака.
4. Входной язык содержит упрощенные условные операторы типа  
   if <логическое выражение> then <оператор присваивания> else <оператор присваивания>; (часть else в операторе может отсутствовать). Логическое выражение может содержать идентификаторы, знаки операций сравнения, целые десятичные числа без знака, скобки и логические операции and и not. Оператор присваивания должен состоять из двух идентификаторов, разделенных знаком присваивания.
5. Входной язык содержит выражения над строковыми константами, разделенные символом ;(точка с запятой). Выражения состоят из идентификаторов, строковых констант, заключенных в двойные кавычки, одиночных символов, заключенных в одинарные кавычки и знаков операции конкатенации +.
6. Входной язык содержит последовательность вызовов процедур, разделенных символом ;(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы, целые десятичные числа без знака, шестнадцатеричные числа, десятичные числа с плавающей точкой.
7. Входной язык содержит последовательность вызовов процедур, разделенных символом ;(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы, строковые константы, заключенные в двойные кавычки и одиночные символы, заключенные в одинарные кавычки.
8. Входной язык содержит последовательность вызовов процедур, разделенных символом ;(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы и римские цифры со знаком. (Римскими считать числа, записанные большими буквами X, V и I)
9. Входной язык содержит последовательность описаний массивов в соответствии со спецификацией языка Паскаль, разделенных символом ;(точка с запятой). Считать, что массивы могут содержать только элементы скалярных типов integer, real, byte, word и char.
10. См. вариант №9, но размер массива должен указываться с помощью римских чисел.
11. Входной язык содержит последовательность описаний записей (record) в соответствии со спецификацией языка Паскаль, разделенных символом ;(точка с запятой). Считать, что записи могут содержать только поля скалярных типов integer, real, byte, word, char и строки string с возможным указанием длины строки в квадратных скобках.
12. Входной язык содержит последовательность описаний констант в соответствии со спецификацией языка Паскаль. Константы могут быть целыми десятичными числами со знаком, целыми шестнадцатеричными числами, целыми десятичными числами с плавающей точкой, строками или символами.
13. Входной язык содержит последовательность команд ассемблера в форме:  
    <метка>: <команда> <операнд1>,<операнд2> (метка, а также один или оба операнда могут отсутствовать). В качестве операндов могут выступать регистры процессора 80х86, идентификаторы, целые десятичные числа или целые шестнадцатеричные числа. (Предусмотреть наличие не менее 6 допустимых команд).
14. Входной язык содержит выражения, разделенные символом; (точка с запятой). Выражения состоят из идентификаторов, бинарных операций (сложение, вычитание, умножение, деление) и функций (например, sqrt, log, exp). Функции могут принимать идентификаторы или числовые константы в качестве аргументов. Также поддерживаются скобки для задания приоритетов операций.
15. Входной язык содержит последовательность логических выражений, разделенных символом; (точка с запятой). Логические выражения могут включать идентификаторы, булевы значения (true, false), знаки операций сравнения (равно, больше, меньше) и логические операции (and, or, not). Кроме того, присутствуют упрощенные условные операторы if <логическое выражение> then <оператор>, где оператор может быть присваиванием или вызовом процедуры.
16. Входной язык содержит описание массивов и записей в виде текстовых строк, разделенных символом; (точка с запятой). Массивы и записи могут включать поля различных скалярных типов: целых чисел, чисел с плавающей точкой, символов и строк. В описании массивов используются только целые числа для указания размеров, в описании записей — различные скалярные типы и строки.
17. Входной язык содержит многоуровневые арифметические выражения, разделенные символом; (точка с запятой). Выражения могут включать идентификаторы, числовые константы, скобки для задания приоритетов, а также функции (например, sin, cos). Операции могут включать сложение, вычитание, умножение и деление.
18. Входной язык содержит цикл for и условные операторы if. Цикл имеет вид for <идентификатор> := <начало> to <конец> do <оператор>, а условный оператор — if <логическое выражение> then <оператор> else <оператор>. В качестве операторов могут быть использованы присваивания или вызовы процедур.
19. Входной язык содержит описание типов данных и констант, разделенных символом; (точка с запятой). Типы данных могут включать простые (целые, вещественные, строковые) и составные (массивы, записи). Константы могут быть целыми числами, строками и символами. Описание типов должно включать указание имен типов и их соответствующих значений.
20. Входной язык содержит описание и вызовы процедур и функций, разделенные символом; (точка с запятой). Описание процедуры или функции включает имя, список параметров и тело. Параметры могут быть идентификаторами, строковыми или числовыми константами. Также предусмотрены вызовы процедур и функций с аргументами, которые могут быть идентификаторами или константами.