**Вариант 1**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.

**Вариант 2**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы, и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do** {/ оператор ";"/ }**end**.

**Вариант 3**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop**непомеченный.

**Вариант 4**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop** {/ оператор ";" /} **end**.

**Вариант 5**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.

**Вариант 6**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do** {/ оператор ";"/ }**end**.

**Вариант 7**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop**непомеченный.

**Вариант 8**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**

**Правила, определяющие идентификатор, букву и цифру**

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop** {/ оператор ";" /} **end**.

**Вариант 9**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do** {/ оператор ";"/ }**end**.

**Вариант 10**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.

**Вариант 11**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (ADD | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop**непомеченный.

**Вариант 12**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /}.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода |

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **инфиксная форма**.

$ присваивание = переменная ASS выражение.

$ выражение = слагаемое { (EQ | NE | LT | GT | LE | GE) слагаемое }.

$ слагаемое = множитель { (PLUS | MIN) множитель }.

$ множитель = унарное { (MUL | DIV | MOD) унарное }.

$ унарное = [ MIN ] терм.

$ терм = переменная | число | "(" выражение")".

$ число = целое | действительное.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

$ условный = **if**выражение**then**непомеченный [**else**непомеченный] ";"

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop** {/ оператор ";" /} **end**.

**Вариант 13**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.

**Вариант 14**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do** {/ оператор ";"/ }**end**.

**Вариант 15**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop**непомеченный.

**Вариант 16**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop** {/ оператор ";" /} **end**

**Вариант 17**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.

**Вариант 18**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.

**Вариант 19**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop**непомеченный.

**Вариант 20**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близкому к языку **Ассемблера:**

$ двоичное = {/ "0" | "1" /} ( "B" | "b" ).

$ восьмеричное = {/ "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" /} ( "C" | "c" ).

$ десятичное = {/ цифра /} [ "D" | "d" ].

$ шестнадцатеричное = цифра {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" } ( "H" | "h" ).

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** {/ оператор ";" /} [ **else** {/ оператор ";" /} ] **end**.

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop** {/ оператор ";" /} **end**.

**Вариант 21**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.

**Вариант 22**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do** {/ оператор ";"/ }**end**.

**Вариант 23**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop**непомеченный.

**Вариант 24**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы различаются.**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - строка десятичных цифр:**

$ метка = имя\_метки ":".

$ имя\_метки = {/ цифра /}.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются **в произвольном стиле**, не используемом в популярных языках программирования:

$ двоичное = "2#" {/ "0" | "1" /}.

$ восьмеричное = "8#" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = ["10#"] {/ цифра /}.

$ шестнадцатеричное = "16#" {цифра | "A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка "." числовая\_строка [порядок] |

числовая\_строка порядок.

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **префиксная форма.**

$ присваивание = ASS переменная "," выражение.

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = операция выражение выражение | "( " MIN выражение ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD |MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **case** выражение **of** целое ":" непомеченный

{ **or** целое ":" непомеченный }

[ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **loop** {/ оператор ";" /} **end**.

**Вариант 25**

Правила, используемые в лексическом анализаторе

Тип лексического анализатора (сканера)

Существуют два основных метода лексического анализа: прямой и непрямой. Они рассматриваются в теме: "Организация лексического анализа". Этот тип лексического анализатора будет использоваться в следующей лабораторной работе для преобразования пользовательского синтаксиса к виду, определяемому заданным значением. Вам необходимо разработать **прямой лексический анализатор**.

Наборы ключевых слов и разделителей

Разрабатываемые языки программирования отличаются используемыми ключевыми словами и разделителями. Вам достался вариант, в котором ключевые слова и разделители заданы в стиле, напоминающем язык программирования **Паскаль.**

*Таблица*

**Назначение специфических ключевых слов и разделителей**

|  |  |  |
| --- | --- | --- |
| **Назначение** | **Обозначение конструкции языка** | **Обозначение лексемы** |
| Сложение | + | PLUS |
| Вычитание | - | MIN |
| Умножение | \* | MULT |
| Деление | / | DIV |
| Остаток по модулю | mod | MOD |
| Равно | = | EQ |
| Не равно | <> | NE |
| Меньше | < | LT |
| Больше | > | GT |
| Меньше или равно | <= | LE |
| Больше или равно | >= | GE |
| Присваивание | := | ASS |
| Начало комментария | { | COMMENT |
| Конец комментария | } |
| Начало составного | begin | BST |
| Конец составного | end | EST |
| Разделитель  Операторов | ; | EOP |

Идентичность прописных и строчных букв (верхнего и нижнего регистров)

Существуют языки программирования, в которых эквивалентные по значению ключевые слова, идентификаторы, метки и другие имена можно задавать с любым сочетанием прописных и строчных букв, так как регистр символа игнорируется. В качестве примера можно назвать Бейсик и Паскаль. В этом случае в блоке лексического анализа имеется специальный модуль, преобразующий поступающие буквы к одному регистру (верхний или нижний регистр выбираются произвольно или с учетом соглашений, принятых в операционной системе при работе с внешними устройствами). В других языках, например Си, Оберон, верхний и нижний регистр различаются. В этом случае регистр ключевых слов отдается на усмотрение разработчика. Дерзайте! Вам достался вариант, в котором **прописные и строчные буквы не различаются**.

Правила, определяющие идентификатор, букву и цифру

$ идентификатор = буква { буква | цифра }.

$ 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".

Организация метки

Метка состоит из ее имени и двоеточия, являющегося ограничителем данной конструкции. Имя метки может быть буквенно-цифровой строкой или целым десятичным числом. Предполагается, что во втором случае имя метки преобразуется в натуральное число в допустимом диапазоне. Поэтому, для одной и той же метки ее представление может отличаться по числу незначащих нулей. Например, метка "1995:" эквивалентна "00001995:". В вашем варианте метка задается следующим образом.

**Метка - буквенно-цифровая строка:**

$ метка = имя\_метки ":".

$ имя\_метки = буква { буква | цифра }.

Организация целых чисел

Предлагается три варианта, отличающихся друг от друга способом представления двоичных, восьмеричных, десятичных и шестнадцатеричных чисел. У Вас они определяются в стиле, близком к языку **Си:**

$ двоичное = "0" ("B" | "b") {/ "0" | "1" /}.

$ восьмеричное = "0" { "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7" }.

$ десятичное = {/ цифра /}.

$ шестнадцатеричное = "0" ("X"|"x") {цифра|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f" /}.

Организация действительных чисел

Действительное число определяется следующим образом:

$ числовая\_строка = {/ цифра /}.

$ порядок = ("E"|"e")["+"|"-"] числовая\_строка.

$ действительное = числовая\_строка порядок |

числовая\_строка "." [числовая\_строка] [порядок] |

"." числовая\_строка [порядок].

Правила, используемые в синтаксическом анализаторе

Рассматриваемые ниже правила используются при построении распознавателя. Альтернативные варианты позволяют задать структуру программы, ее операторов и выражений. Ключевые слова при описании правил выделены жирным шрифтом. Их реальное написание определяется совокупностью условий и ограничений, определяемых вариантом задания (прописные или строчные и т.д.)

Организация программы

Задается один из двух вариантов. Первый вариант определяет программу как список чередующихся описаний и операторов, разделяемых точкой с запятой. Конец текста программы определяется концом файла. Описания можно вводить непосредственно перед использованием переменных. При втором варианте программа состоит из двух независимых областей: описаний и операторов. За лексемой, определяющей конец программы, может следовать произвольная цепочка символов, так как осуществлять разбор дальше в соответствии с синтаксисом не имеет смысла. С организацией программы непосредственно связывается и структура составного оператора, синтаксис которого выдержан в аналогичном стиле. Вот ваш вариант этих конструкций.

$ программа = {/ (описание | оператор) ";" /} конец\_файла.

$ составной = BST {/ оператор ";" /} EST.

Описания

Первый вариант описания по стилю близок к языку программирования Паскаль, а второй напоминает описания Си. Вот тот, который выпал Вам.

$ описание = идентификатор { "," идентификатор } ":" [ **vector** "[" целое "]" **of** ] тип.

Правило, определяющее тип, является общим для обоих вариантов:

$ тип = integer | real.

Синтаксис операторов

Состав операторов является одинаковым для всех вариантов. Различия наблюдаются при описании их синтаксиса. Одинаковым для всех является синтаксис таких операторов, как пустой, перехода, ввода и вывода. Следует также отметить, что пустой оператор - это отсутствие каких бы то ни было конструкций, а не точка, которая в данном случае является компонентой метаязыка, завершающей правило. Кроме операторов здесь же приведен еще ряд конструкций языка, являющихся общими для всех вариантов.

$ оператор = [метка] непомеченный.

$ непомеченный = составной | присваивания | перехода

условный | цикла | пустой | ввода | вывода.

$ пустой = .

$ перехода = **goto**имя\_метки.

$ ввода = **read** переменная { "," переменная }.

$ вывода = **write**( выражение | спецификатор )

{ "," ( выражение | спецификатор ) }.

$ переменная = идентификатор [ "[" индекс "]" ].

$ индекс = идентификатор | целое.

$ спецификатор = **skip** | **space** | **tab**.

Оператор присваивания

Определяется вместе с выражением, которое ему присваивается, что обеспечивает согласованное восприятие языковых конструкций. Выражение может использоваться и в других операторах языка, например в операторе вывода. Существует три альтернативы для оператора присваивания и выражений: инфиксная форма, постфиксная скобочная форма и польская префиксная форма. Вам досталась **постфиксная форма**.

$ присваивание = выражение переменная ASS .

$ выражение = простое\_выражение | операнд.

$ простое\_выражение = "(" выражение выражение операция ")" | "(" выражение MIN ")".

$ операнд = переменная | целое | действительное.

$ операция = MUL | DIV | MOD | ADD | MIN | EQ | NE | LT | GT | LE | GE.

Условный оператор

Альтернативы подобраны по стилю, обеспечивающему совпадение с синтаксисом программы и определяют обычный условный оператор или переключатель, обеспечивающий несколько ветвлений.

$ условный = **if** выражение **then** непомеченный [ **else** непомеченный ].

Оператор цикла

Синтаксис альтернатив в вариантах подобран в стиле программы, как и в случае с условными операторами.

$ цикла = **while** выражение **do**непомеченный.