Лабораторная работа №5 (Вариант II)

Основы лексического и синтакисческого анализа

Цель работы

Получение навыков реализации лексических анализаторов и нисходящих синтаксических анализаторов, использующих метод рекурсивного спуска.

Вопросы для допуска к работе

1. Дайте определение понятию «грамматика». В его контексте определите понятия «терминиальный символ», «нетерминальный символ», «правило грамматики».

2. Какая из приведенных ниже грамматик является LL(k)-грамматикой? Можно ли непосредственно на основе этой грамматики реализовать нисходящий рекурсивный парсер? Почему? Приведите пример исходной последовательности для этой грамматики.

   - $A \to x\,B\,y$
$B \to A\,C$
$C \to z\,B \mid A$

   - $A \to B \mid x\,y$
$B \to x\,t\,y$

3. Какой вычислительный процесс — линейно- или древовидно-рекурсивный — имеет место в случае: (а) синтаксического анализа, (б) лексического анализа? Почему?

Задания

1. Реализуйте простейшие сканеры:

   - Процедуру `check-frac`, принимающую на вход строку и возвращающую `#t`, если в строке записана простая дробь формате *десятичное-целое-со-знаком/десятичное-целое-без-знака*, и `#f` в противном случае. Запрещается применять встроенную процедуру для преобразования строки к числу.

   - Процедуру `scan-frac`, принимающую на вход строку и возвращающую `#t`, если в строке записана простая дробь формате *десятичное-целое-со-знаком/десятичное-целое-без-знака*, и `#f` в противном случае. Запрещается применять встроенные процедуры преобразования строки к числу к исходной строке до её посимвольной обработки сканером.

   - Процедуру `scan-many-fracs`, принимающую на вход строку, содержащую простые дроби, разделенные пробельными символами (строка также может начинаться и заканчиваться произвольным числом пробелов, символов табуляции, перевода строки и др.), и возвращающую список этих дробей. Если

разбор не возможен, процедура должна возвращать `#f`. Запрещается применять встроенные процедуры преобразования строки к числу к исходной строке до её посимвольной обработки сканером.

Примеры вызова процедур:

```
(check-frac "110/111") ⇒ #t
(check-frac "-4/3")    ⇒ #t
(check-frac "+5/10")   ⇒ #t
(check-frac "5.0/10")  ⇒ #f
(check-frac "FF/10")   ⇒ #f

(check-frac "110/111") ⇒ 110/111
(check-frac "-4/3")    ⇒ -4/3
(check-frac "+5/10")   ⇒ 5/10
(check-frac "5.0/10")  ⇒ #f
(check-frac "FF/10")   ⇒ #f

(scan-many-fracs
 "\t1/2 1/3\n\n10/8") ⇒ (1/2 1/3 10/8)
(scan-many-fracs
 "\t1/2 1/3\n\n2/-5")    ⇒ #f
```

В начале текста программы, в комментариях, обязательно запишите грамматику в БНФ или РБНФ, которую реализуют ваши сканеры.

Рекомендация. Символ, маркирующий конец последовательности, выберете исходя из того, что на вход вашего лексера может поступить любая последовательность символов из таблицы ASCII, встречающаяся в текстовых файлах.

2. Реализуйте процедуру `parse`, осуществляющую разбор программы на модельном языке, представленную в виде последовательности (вектора) токенов (см. Лабораторную работу №4 «Интерпретатор стекового языка программирования»). Процедура `parse` должна включать в себя реализацию синтаксического анализа последовательности токенов методом рекурсивного спуска согласно следующей грамматике:

```
<Program>  ::= <Articles> <Body> .
<Articles> ::= <Article> <Articles> | .
<Article>  ::= define word <Body> end .
<Body>     ::= if <Body> endif <Body> | integer <Body> | word <Body> | .
```

Процедура должна возвращать синтаксическое дерево в виде вложенных списков, соответствующих нетерминалам грамматики. В случае несоответствия входной последовательности грамматике процедура должна возвращать `#f`. Примеры применения процедуры:

```
(parse #(1 2 +)) ⇒ (() (1 2 +))

(parse #(x dup 0 swap if drop -1 endif))
    ⇒ (() (x dup 0 swap (if (drop -1))))

(parse #( define -- 1 - end
          define =0? dup 0 = end
          define =1? dup 1 = end
          define factorial
              =0? if drop 1 exit endif
              =1? if drop 1 exit endif
              dup --
              factorial
              *
```

```
          end
          0 factorial
          1 factorial
          2 factorial
          3 factorial
          4 factorial ))
 ⇒
 (((-- (1 -))
   (=0? (dup 0 =))
   (=1? (dup 1 =))
   (factorial
    (=0? (if (drop 1 exit)) =1? (if (drop 1 exit)) dup -- factorial *)))
  (0 factorial 1 factorial 2 factorial 3 factorial 4 factorial))

(parse #(define word w1 w2 w3)) ⇒ #f
```

Подготовьте еще 2-3 примера для демонстрации. Обратите внимание, что грамматика позволяет записывать на исходном языке вложенные конструкции `if` .. `endif`. Учтите эту особенность при реализации парсера и продемонстрируйте её на примерах.

Как изменится грамматика, если допустить вложенные статьи?