Лабораторная 2: ручное построение нисходящих парсеров

Вариант: 8 (Описание лямбда функций в Python)

Этот документ автоматически форматируется в Okular, но на всякий случай я экспортировал его заранее в README.pdf

Задание 1: Грамматика

Для наглядности, терминалы пишутся не так как называются токены (i.e. ")" вместо RPAREN, и "lambda" вместо `LAMBDA_KW`, `var` вместо `VARIABLE`)

Исходная грамматика

```
Declaration -> "lambda" Arglist ":" Expression

Arglist -> ""
Arglist -> Varlist
Varlist -> Varlist "," var
Varlist -> var

Expression -> Expression + Term
Expression -> Term
Term -> Term * Factor
Term -> Factor
Factor -> (E)
Factor -> var
```

Описание нетерминалов

```
Нетерминал      Описание
Declaration     Описание лямбда функции в Python
Arglist         Список аргументов функции (может быть пустым)
Varlist         Непустой список comma-separated переменных
Expression      Арифметическое выражение с переменными (без чисел, с операциями + и
                *)
Term            Терм в арифметическом выражении
Factor          Множитель в арифметическом выражении
```

Модифицированная грамматика

В исходной грамматике есть левая рекурсия, избавимся от неё:

```
Declaration -> "lambda" Arglist ":" Expression

Arglist -> Varlist
Arglist -> ""
~ Varlist -> var Varlist'
~ Varlist' -> "," var Varlist'
+ Varlist' -> eps

~ Expression -> Term Expression'
~ Expression' -> "+" Term Expression'
+ Expression' -> ""
~ Term -> Factor Term'
~ Term' -> "*" Factor Term'
+ Term' -> ""
Factor -> (E)
Factor -> var
```

(~ - измененная строка, + - новая строка, по аналогии с `diff`)

Описание нетерминалов

Запись `-//-` значит "ditto":

```
Нетерминал      Описание
Declaration     -//-
```

```
Arglist         -//-
Varlist         -//-
Varlist'        Продолжение списка из переменных
Expression      -//-
Expression'     Продолжение арифметического выражения
Term            -//-
Term'           Продолжение терма (т.е. операнда суммирования)
Factor          -//-
```

Задание 2: Лексический анализатор

```
Терминал    Токен
"("         LPAREN
")"         RPAREN
$           END
"+"         PLUS
"*"         ASTERISK
var         VARIABLE
"lambda"    LAMBDA_KW
":"         COLON
","         COMMA
```

Задание 3: Синтаксический анализатор

Таблица FIRST и FOLLOW исходной грамматики

```
Нетерминал       FIRST        FOLLOW
Declaration      "lambda"     $
Arglist          var eps      ":"
Varlist          var          ":" ","
Expression       "(" var      $ "+" ")"
Term             "(" var      $ "*" ")"
Factor           "(" var      $ ")"
```

Таблица FIRST и FOLLOW модифицированной грамматики

```
Нетерминал     FIRST       FOLLOW
Declaration    "lambda"    $
Arglist        var eps     ":"
Varlist        var         ":"
Varlist'       "," eps     ":"
Expression     "(" var     $ ")"
Expression'    "+" eps     $ ")"
Term           "(" var     $ "+" ")"
Term'          "*" eps     $ "+" ")"
Factor         "(" var     $ "*" "+" ")"
```

TODO

Задание 4: Визуализация дерева разбора

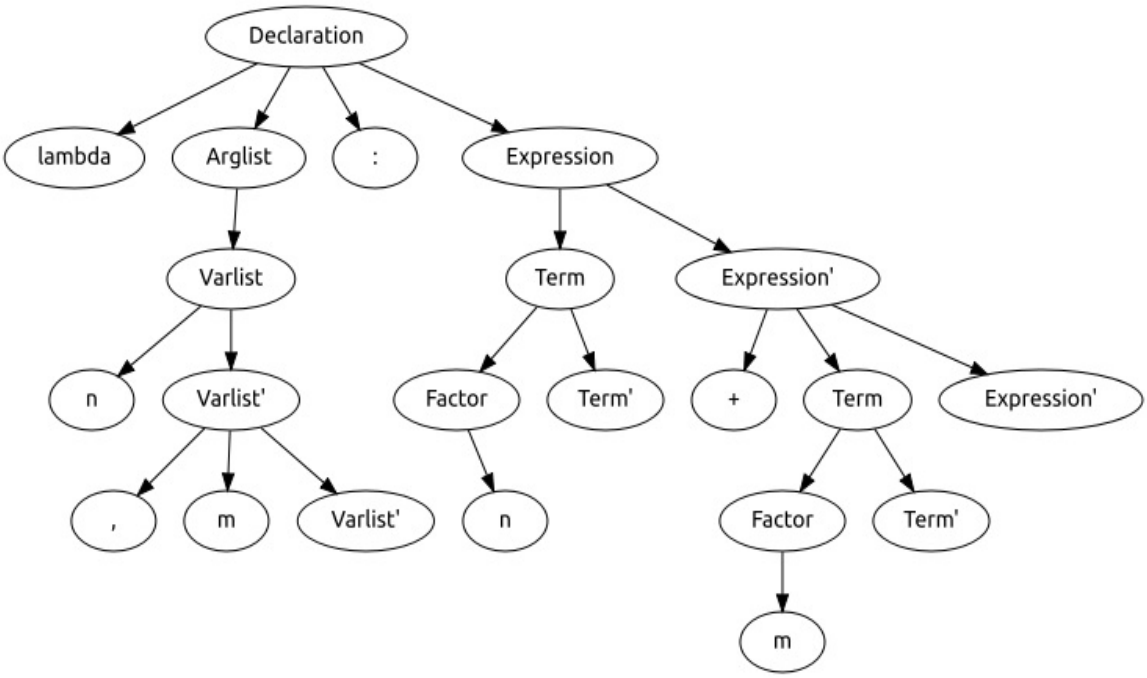
Пример использования:

```
$ ./parser >pic.dot && dot -T svg pic.dot >pic.svg && open pic.svg
lambda n, m: n + m
<ctrl+d to indicate end of input>
```

Либо

```
$ make picture
lambda n, m: n + m
<ctrl+d to indicate end of input>
```

Получившаяся картинка:



Задание 5: Тесты

Автоматические тесты

```
make test
```

Валидные строки

Программа должна завершиться без ошибки и вывести AST для graphviz:

Вход	Вывод
`lambda: n`	OK
`lambda  :    n`	OK
`lambda x: x*x`	OK
`lambda x   : x  *x`	OK
`lambda a, b:   a + b * (c)`	OK
`lambda x,y,z: (((x*x + y*y + z*z)))`	OK
`lambda x,lambdaa: x + lambdaa`	OK

(Полный список тестов лежит в `./examples`)

Невалидные строки

Программа должна завершиться с ошибкой и вывести соответствующее сообщение:

Вход	Вывод программы (stderr)
`lambda`	`Expected argument list, got `` at position 7`
`lambda:`	`Expected beginning of expression, got `` at position 8`
`lambda n : :`	`Expected beginning of expression, got `:` at position 12`
`: n`	`Expected lambda declaration, got `:` at position 1`
`;;; invalid token chars`	`There are no tokens that start with `;` (at pos 1)`
`a bunch of tokens`	`Expected lambda declaration, got `a` at position 1`
`lambda varlist without commas: x`	`Expected continuation of variable list, got `without` at position 16`

(полный список тестов лежит в `./invalid-examples`)