Государственное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»

Отчет
По лабораторной работе
По курсу «Конструирование компиляторов»
На тему
«Синтаксический анализатор операторного предшествования»

Студент: Горин Д.И.
Группа: ИУ7-23М
Вариант: 3
Преподаватель: Ступников А.А.

Москва, 2020

Оглавление

1 Цель и задачи работы

Цель работы: приобретение практических навыков реализации таблично управляемых синтаксических анализаторов на примере анализатора операторного предшествования

Задачи работы:

1. Ознакомиться с основными понятиями и определениями, лежащими в основе синтаксического анализа операторного предшествования.
2. Изучить алгоритм синтаксического анализа операторного предшествования.
3. Разработать, тестировать и отладить программу синтаксического анализа в соответствии с предложенным вариантом грамматики.
4. Включить в программу синтаксического анализ семантические действия для реализации синтаксически управляемого перевода инфиксного выражения в обратную польскую нотацию.

2 Листинг

2.1 main.py

```
1  from analyzer import analyze_string
2
3
4  if __name__ == '__main__':
5
6      input_str = '1 < 2'
7      ans = analyze_string(input_str)
8      print(f'Ans is {ans}, should be: 12<')
9
10     input_str = '3 <> 4'
11     ans = analyze_string(input_str)
12     print(f'Ans is {ans}, should be: 34<>')
13
14     input_str = '1 + 2 < 3 - 4'
15     ans = analyze_string(input_str)
16     print(f'Ans is {ans}, should be: 12+34-<')
17
18     input_str = '1 + 2 < 3 + 4 / 5'
19     ans = analyze_string(input_str)
20     print(f'Ans is {ans}, should be: 12+345/+<')
21
22     input_str = '1 + 2 < (3 + 4) / 5'
23     ans = analyze_string(input_str)
24     print(f'Ans is {ans}, should be: 12+34+5/<')
25
26     input_str = '(1 <> 3) < (2 + 3) / 4'
27     ans = analyze_string(input_str)
28     print(f'Ans is {ans}, should be: 13<>23+4/<')
```

2.2 analyzer.py

```
1  from typing import Tuple, List, Optional, Union
2
3
4  def _get_next_symbol(input_str: str) -> Tuple[str, str]:
5      """Получениеследующегосимволавстрокесимволы
6
7       >= <= == <> выдаютсякакодинсимвол
8      """
9      cur_symbol = input_str[0]
10     if cur_symbol in ('>', '=') and len(input_str) > 1 and input_str
           [1] == '=':
11         cur_symbol += input_str[1]
12         input_str = input_str[2:]
13     elif cur_symbol == '<' and len(input_str) > 1 and input_str[1]
           in ('=', '>'):
14         cur_symbol += input_str[1]
15         input_str = input_str[2:]
16     else:
17         input_str = input_str[1:]
18     return input_str, cur_symbol
19
20
21 def _get_priority(stack_symbol: str, next_symbol: str) -> Optional[
       str]:
22     """Приоритеты
23
24     """
25     sum_ops = ('+', '-')
26     mul_ops = ('*', '/', '%', '^')
27     rel_ops = ('<', '<=', '>', '>=', '<>', '==')
28     numbers = tuple(str(i) for i in range(10))
29
30     if stack_symbol in sum_ops:
31         return '>' if next_symbol in sum_ops + rel_ops + (')', '$')
               else '<'
32     if stack_symbol in mul_ops:
33         return '>' if next_symbol in sum_ops + mul_ops + rel_ops + (
               ')', '$') else '<'
34     if stack_symbol in rel_ops:
35         return '>' if next_symbol in rel_ops + (')', '$') else '<'
36     if stack_symbol in numbers:
37         return '>' if next_symbol in sum_ops + mul_ops + rel_ops + (
               ')', '$') else None
38     if stack_symbol == '(':
39         return '<' if next_symbol in sum_ops + mul_ops + rel_ops +
               numbers + ('(', ')') else None
40     if stack_symbol == ')':
41         return '>' if next_symbol in sum_ops + mul_ops + rel_ops + (
```

```
            ')', '$') else None
42      if stack_symbol == '$':
43          return '<' if next_symbol in sum_ops + mul_ops + rel_ops +
               numbers + ('(', ) else None
44      return None
45
46
47  def _get_nterm_of_rule(stack_slice: List) -> Optional[str]:
48      """Получаемлевуючастьправилапоправой
49
50      """
51      sum_ops = ('+', '-')
52      mul_ops = ('*', '/', '%', '^')
53      rel_ops = ('<', '<=', '>', '>=', '<>', '==')
54      numbers = tuple(str(i) for i in range(10))
55      if len(stack_slice) == 3:
56          if ''.join(stack_slice) == '(E)':
57              return 'E'
58          if stack_slice[0] == 'E' and stack_slice[2] == 'E' and
               stack_slice[1] in sum_ops + mul_ops + rel_ops:
59              return 'E'
60      elif len(stack_slice) == 1:
61          if stack_slice[0] in numbers:
62              return 'E'
63      return None
64
65
66  def analyze_string(input_str: str) -> Optional[str]:
67      """ПСанализатор
68      -
69      """
70      numbers = tuple(str(i) for i in range(10))
71      numbers_stack = []
72      input_str = ''.join([symbol for symbol in input_str if symbol
           not in [' ', '\n']]) + '$'
73      stack = ['$']
74      postfix = ''
75      while stack != ['$', 'E'] or input_str != '$':
76          # Получаемследующийсимволвстроке
77          input_str, cur_symbol = _get_next_symbol(input_str)
78          if cur_symbol in numbers:
79              numbers_stack.append(cur_symbol)
80
81          # Получаемприоритет
82          last_term = [x for x in stack if x != 'E'][-1]
83          pr = _get_priority(last_term, cur_symbol)
84          if pr is None:
85              return None
86
87          if pr == '>':
```

```
88                input_str = cur_symbol + input_str
89                # Свертка
90                for i in range(1, len(stack)):
91                    stack_slice = stack[-i:]
92                    lhs_nterm = _get_nterm_of_rule(stack_slice)
93                    if lhs_nterm is not None:
94                        stack = stack[:-i]
95                        stack.append(lhs_nterm)
96                        if len(stack_slice) == 3 and stack_slice[0] != '
                              (':
97                            while len(numbers_stack) > 0:
98                                postfix += numbers_stack.pop(0)
99                            postfix += stack_slice[1]
100                       break
101               else:
102                   return None
103           else:
104               # Укладкавстек
105               stack.append(cur_symbol)
106   return postfix
```

3 Тесты

1. $1 < 2 \rightarrow 12 <$
2. $3 <> 4 \rightarrow 34 <>$
3. $1 + 2 < 3 - 4 \rightarrow 12 + 34 - <$
4. $1 + 2 < 3 + 4/5 \rightarrow 12 + 345/ + <$
5. $1 + 2 < (3 + 4)/5 \rightarrow 12 + 34 + 5/ <$
6. $(1 <> 3) < (2 + 3)/4 \rightarrow 13 <> 23 + 4/ <$

4 Выводы

По результатам проведенной работы студент приобрел практические навыки в реализации алгоритма синтаксического анализатора операторного предшествования.

5 Список литературы

1. БЕЛОУСОВ А.И., ТКАЧЕВ С.Б. Дискретная математика: Учеб. Для вузов / Под ред. В.С. Зарубина, А.П. Крищенко. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.

2. АХО А., УЛЬМАН Дж. Теория синтаксического анализа, перевода и компиляции: В 2-х томах. Т.1.: Синтаксичечкий анализ. - М.: Мир, 1978.

3. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.