|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления ​​​

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии ​​​

**Отчет по лабораторной работе**

**«Синтаксический анализатор операторного предшествования»**

**по курсу «Конструирование компиляторов»**

**Вариант 5**

Выполнил студент группы ИУ7-21М Доманов К. И.

Проверил Ступников А. А.

*2022 г.*

**Описание задания**

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

В процессе выполнения лабораторной работы в соответствии с вариантом 4, необходимо реализовать синтаксический анализатор операторного предшествования и синтаксически управляемый перевод инфиксного выражения в обратную польскую нотацию для грамматики выражений из лабораторной работы №4.

**Теоретическая часть**

Метод операторного предшествования относится к классу восходящих таблично-управляемых методов синтаксического анализа на основе алгоритма типа «перенос/свертка». Этот метод основан на отношениях предшествования Вирта-Вебера, но только между терминальными символами грамматики.

Операторной грамматикой называется приведенная КС-грамматика без ε-правил, в которой правые части правил не содержат смежных нетерминалов.

Операторная грамматика G называется грамматикой операторного предшествования, если между любыми двумя терминальными символами выполняется не более одного отношения операторного предшествования.

**Исходные данные**

Матрица отношений операторного предшествования будет иметь вид:

"a": {  
 "a": '0', "+": '>', "-": '>', "\*\*": '>', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '0', ")": '>', "$": '>'  
},  
"+": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '<', "/": '<', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"-": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '<', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"\*\*": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '>', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"\*": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"/": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"<": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
">": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"and": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"or": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '<', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
},  
"(": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '<', "or": '<',  
 "(": '<', ")": '=', "$": '0'  
},  
")": {  
 "a": '0', "+": '>', "-": '>', "\*\*": '>', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '0', ")": '>', "$": '>'  
},  
"$": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '<', "or": '<',  
 "(": '<', ")": '0', "$": '1'  
}

Необходимо реализовать синтаксический анализатор операторного предшествования для данной матрицы отношений.

**Построение анализатора операторного предшествования**

В ходе лабораторной работы, был реализован синтаксический анализатор операторного предшествования. Результат работы программы представлен на рисунках 1 – 3.

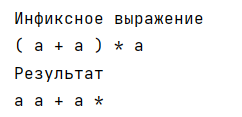


Рисунок 1 – Результат работы программы

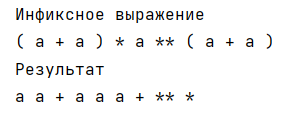


Рисунок 2 – Результат работы программы

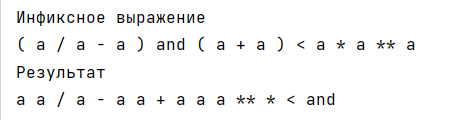


Рисунок 3 – Результат работы программы

**Текст программы**

relation\_table = {  
 "a": {  
 "a": '0', "+": '>', "-": '>', "\*\*": '>', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '0', ")": '>', "$": '>'  
 },  
 "+": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '<', "/": '<', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "-": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '<', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "\*\*": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '>', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "\*": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "/": {  
 "a": '<', "+": '>', "-": '>', "\*\*": '<', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "<": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 ">": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "and": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '>', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "or": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '<', "or": '>',  
 "(": '<', ")": '>', "$": '>'  
 },  
 "(": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '<', "or": '<',  
 "(": '<', ")": '=', "$": '0'  
 },  
 ")": {  
 "a": '0', "+": '>', "-": '>', "\*\*": '>', "\*": '>', "/": '>', "<": '>', ">": '>', "and": '>', "or": '>',  
 "(": '0', ")": '>', "$": '>'  
 },  
 "$": {  
 "a": '<', "+": '<', "-": '<', "\*\*": '<', "\*": '<', "/": '<', "<": '<', ">": '<', "and": '<', "or": '<',  
 "(": '<', ")": '0', "$": '1'  
 }  
}  
  
stack = ['$']  
reverse\_polish\_notation = []  
  
  
def parser(input\_string):  
 for i in range(0, input\_string.\_\_len\_\_()):  
 if i == input\_string.\_\_len\_\_() - 1:  
 break  
 if input\_string[i] in ('+', '-', '\*', '/', '\*\*', '<', '>', 'and', 'or') and input\_string[i + 1] in (  
 '+', '-', '\*', '/', '\*\*', '<', '>', 'and', 'or'):  
 print('Операторная грамматика G называется грамматикой операторного предшествования,\nесли между любыми '  
 'двумя терминальными символами выполняется не более одного\nотношения операторного предшествования!')  
 exit()  
 i = 0  
 is\_error = False  
 while stack.\_\_len\_\_() > 1 or input\_string[i] != '$':  
 if relation\_table[stack[-1]][input\_string[i]] in ('=', '<'):  
 *# Перенос* stack.append(input\_string[i])  
 i = i + 1  
 elif relation\_table[stack[-1]][input\_string[i]] == '>':  
 *# Свертка* while True:  
 term = stack.pop()  
 if term not in ('(', ')'):  
 reverse\_polish\_notation.append(term)  
 if relation\_table[stack[-1]][term] == '<':  
 break  
 elif relation\_table[stack[-1]][input\_string[i]] == '0':  
 print('Недопустимая последовательность символов')  
 exit()  
  
 out = ' '.join([str(item) for item in reverse\_polish\_notation])  
 if not is\_error:  
 print('Результат')  
 print(out)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# input\_string = '( a + a ) \* a'  
 # input\_string = '( a + a ) \* a \*\* ( a + a )'  
 # input\_string = 'a < ( a + a ) \* a \*\* ( a + a )'* input\_string = '( a / a - a ) and ( a + a ) < a \* a \*\* a'  
 *# input\_string = '( a / a - a ) + ( a + a ) - a \* a \*\* a'  
 # input\_string = ' ( a + ( a ) \*\* ( a / a ) ) '  
 # input\_string = ' a + a'* print('Инфиксное выражение')  
 print(input\_string)  
 input\_string = input\_string + ' $'  
 input\_string = list(input\_string.strip().split())  
 parser(input\_string)