Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ

по лабораторной работе

на тему

Синтаксический анализ

Выполнил

Студент гр. 053501

Хиль В.М.

Проверил

Ассистент кафедры информатики

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc131884964)

[2 Краткие теоретические сведения 4](#_Toc131884965)

[3 Примеры работы синтаксического анализатора 5](#_Toc131884966)

[3.1 Работа анализатора 5](#_Toc131884970)

[3.2 Синтаксические ошибки 7](#_Toc131884971)

[Приложение А (обязательное) Листинг кода 10](#_Toc131884972)

[Приложение Б (обязательное) Узлы дерева 12](#_Toc131884973)

[Приложение В (обязательное) Классы для обнаружения ошибок 16](#_Toc131884974)

**1 ЦЕЛЬ РАБОТЫ**

Освоение работы с существующими синтаксическими анализаторами. Разработать свой собственный синтаксический анализатор, выбранного подмножества языка программирования. Построить синтаксическое дерево. Определить минимум 4 возможных синтаксических ошибки и показать их корректное выявление.

1. **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

В ходе синтаксического анализа исходный текст программы проверяется на на соответствие синтаксическим нормам языка с построением дерево разбора (синтаксическое дерево), которое отражает синтаксическую структуру входной последовательности и удобно для дальнейшего использования, а также в случае несоответствия – позволяет вывести сообщения об ошибках.

Как правило, результатом синтаксического анализа является синтаксическое строение предложения, представленное либо в виде дерева зависимостей, либо в виде дерева составляющих, либо в виде некоторого сочетания первого и второго способов представления.

Таким образом на основе анализа выражений, состоящих из литералов, операторов и круглых скобок выполняется группирование токенов исходной программы в грамматические фразы, используемые для синтеза вывода. Представление грамматических фраз исходной программы выполнить в виде дерева. Реализовать синтаксический анализатор с использованием одного из табличных методов (LL-, LR-метод, метод предшествования и пр.).

Т.е. мы имеем последовательность шагов в виде помеченного дерева. Внутренние вершины представляют те действия, которые можно выполнять. Прямые потомки каждой вершины либо представляют аргументы, к которым нужно применять действие (если соответствующая вершина помечена идентификатором или является внутренней), либо помогают определить, каким должно быть это действие, в частности, знаки «+», «\*» и «=». Скобки отсутствуют, т.к. они только определяют порядок действий.

LL- и LR- методы позволят обнаружить ошибки на самых ранних стадиях, т.е. когда разбор потока токенов от лексического анализатора в соответствии с грамматикой языка становится невозможен. Можно использовать нисходящий (англ. top-down parser) со стартового символа, до получения требуемой последовательности токенов. Для этих целей применим метод рекурсивного спуска либо LL-анализатор. Или использовать восходящий (англ. bottom-up parser) - продукции восстанавливаются из правых частей, начиная с токенов и кончая стартовым символом - LR-анализатор и прочее.

1. **ПРИМЕРЫ РАБОТЫ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА**

Текущая версия программы передает список токенов и файл, содержащий грамматику синтаксическому анализатору, который согласно правилам, описанным там, с помощью LR-алгоритма формирует синтаксическое дерево.

3. 1. **Работа анализатора**

Для примера возьмем следующую программу (см. рисунок 1):

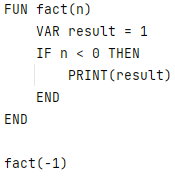


Рисунок 1 – Входные данные

В результате было построено синтаксическое дерево (см. рисунок 2), состоящее из различных синтаксических узлов программы (см. приложение Б):





Рисунок 2 – Результат работы синтаксического анализатора

* 1. **Синтаксические ошибки**

Синтаксические ошибки возникают при выполнении синтаксического анализа и обычно указывают на то, что в коде есть неправильно написанные или расставленные элементы. Чтобы исправить синтаксические ошибки, нужно провести проверку на то, что все элементы расставлены правильно и соответствуют правилам выбранного подмножества языка.

Ошибка нарушения баланса скобок представлена на рисунке 3.

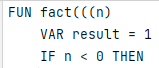


Рисунок 3 – Программа с ошибкой

На рисунке 4 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

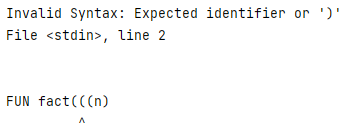


Рисунок 4 – Результат работы синтаксического анализатора

Ошибка неполного объявления переменной представлена на рисунке 5.

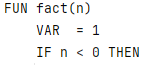


Рисунок 5 – Программа с ошибкой

На рисунке 6 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

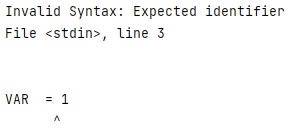


Рисунок 6 – Результат работы синтаксического анализатора

Ошибка неожиданного токена представлена на рисунке 7.

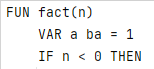


Рисунок 7 – Программа с ошибкой

На рисунке 8 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

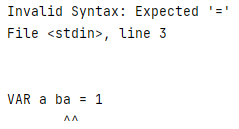


Рисунок 8 – Результат работы синтаксического анализатора

Ошибка неправильной последовательности токенов представлена на рисунке 9.

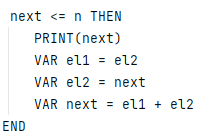


Рисунок 9 – Программа с ошибкой

На рисунке 10 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

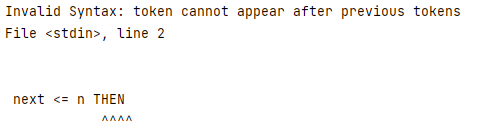


Рисунок 10 – Результат работы синтаксического анализатора

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

class Parser:  
 def \_\_init\_\_(self, tokens):  
 self.tokens = tokens  
 self.token\_index = -1  
 self.advance()  
  
 def update\_current\_token(self):  
 if 0 <= self.token\_index < len(self.tokens):  
 self.current\_token = self.tokens[self.token\_index]  
  
 def advance(self):  
 self.token\_index += 1  
 self.update\_current\_token()  
 return self.current\_token  
  
 def reverse(self, amount=1):  
 self.token\_index -= amount  
 self.update\_current\_token()  
 return self.current\_token  
  
 def parse(self):  
 result = self.statements()  
 if not result.error and self.current\_token.type != EOF:  
 return result.failure(InvalidSyntaxError(  
 self.current\_token.start\_position, self.current\_token.end\_position,  
 "token cannot appear after previous tokens"  
 ))  
 return result

class ParseResult:  
 def \_\_init\_\_(self):  
 self.error = None  
 self.node = None  
 self.last\_registered\_count\_advance = 0  
 self.count\_advance = 0  
 self.to\_reverse\_count = 0  
  
 def register(self, result):  
 self.last\_registered\_count\_advance = result.count\_advance  
 self.count\_advance += result.count\_advance  
 if result.error: self.error = result.error  
 return result.node  
  
 def try\_to\_register(self, result):  
 if result.error:  
 self.to\_reverse\_count = result.count\_advance  
 return None  
 return self.register(result)  
  
 def success(self, node):  
 self.node = node  
 return self  
  
 def failure(self, error):  
 if not self.error or self.last\_registered\_count\_advance == 0:  
 self.error = error  
 return self  
  
 def register\_advancement(self):  
 self.last\_registered\_count\_advance = 1  
 self.count\_advance += 1

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Узлы дерева**

class ListNode:  
 def \_\_init\_\_(self, element\_nodes, start\_position, end\_position):  
 self.element\_nodes = element\_nodes  
  
 self.start\_position = start\_position  
 self.end\_position = end\_position  
  
 def \_\_repr\_\_(self):  
 return f'\n\033[91mListNodes:{self.element\_nodes}\033[91m'  
  
  
class NumberNode:  
 def \_\_init\_\_(self, token):  
 self.token = token  
  
 self.start\_position = self.token.start\_position  
 self.end\_position = self.token.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[31mType:{self.token.type}, (\033[31mValue:{self.token.value}\033[31m)\033[31m)'  
  
  
class StringNode:  
 def \_\_init\_\_(self, token):  
 self.token = token  
  
 self.start\_position = self.token.start\_position  
 self.end\_position = self.token.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[31mType:{self.token.type}, (\033[31mValue:{self.token.value}\033[31m)\033[31m)'  
  
  
class BinaryOperationNode:  
 def \_\_init\_\_(self, left\_node, operation\_token, right\_node):  
 self.left\_node = left\_node  
 self.operation\_token = operation\_token  
 self.right\_node = right\_node  
  
 self.start\_position = self.left\_node.start\_position  
 self.end\_position = self.right\_node.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[35mBinOpNode:(\033[35mLeftNode:{self.left\_node}, (\033[35mOperation:{self.operation\_token},' \  
 f'(\033[35mRightNode:{self.right\_node}\033[35m)\033[35m)\033[35m)\033[35m)'  
  
  
class UnaryOperationNode:  
 def \_\_init\_\_(self, operation\_token, node):  
 self.operation\_token = operation\_token  
 self.node = node  
  
 self.start\_position = self.operation\_token.start\_position  
 self.end\_position = node.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[35mUnaryOpNode:(\033[35mOperation:{self.operation\_token}, (\033[35mNodeUnary:{self.node}' \  
 f'\033[35m)\033[35m)\033[35m)'  
  
  
class VarAccessNode:  
 def \_\_init\_\_(self, var\_name\_token):  
 self.var\_name\_token = var\_name\_token  
  
 self.start\_position = self.var\_name\_token.start\_position  
 self.end\_position = self.var\_name\_token.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[33mVarAccessNode:(\033[33m{self.var\_name\_token}\033[33m)\033[33m)'  
  
  
class VarAssignNode:  
 def \_\_init\_\_(self, var\_name\_token, value\_node):  
 self.var\_name\_token = var\_name\_token  
 self.value\_node = value\_node  
  
 self.start\_position = self.var\_name\_token.start\_position  
 self.end\_position = self.value\_node.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[34mVarAssignNode:(\033[34mTypeVar:{self.var\_name\_token}, (\033[34mValue:{self.value\_node}' \  
 f'\033[34m)\033[34m)\033[34m)'  
  
  
class IfNode:  
 def \_\_init\_\_(self, cases, else\_case):  
 self.cases = cases  
 self.else\_case = else\_case  
  
 self.start\_position = self.cases[0][0].start\_position  
 self.end\_position = (self.else\_case or self.cases[len(self.cases) - 1])[0].end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[36mIfNode:(\033[36mCases:{self.cases}, (\033[36mElse:{self.else\_case}\033[36m)\033[36m)\033[36m)'  
  
  
class ForNode:  
 def \_\_init\_\_(self, var\_name\_token, start\_value\_node, end\_value\_node, step\_value\_node, body\_node, should\_return\_null):  
 self.var\_name\_token = var\_name\_token  
 self.start\_value\_node = start\_value\_node  
 self.end\_value\_node = end\_value\_node  
 self.step\_value\_node = step\_value\_node  
 self.body\_node = body\_node  
 self.should\_return\_null = should\_return\_null  
  
 self.start\_position = self.var\_name\_token.start\_position  
 self.end\_position = self.body\_node.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[37mForNode:(\033[37mVarName:{self.var\_name\_token}, ' \  
 f'(\033[37mStartValue:{self.start\_value\_node}, ' \  
 f'(\033[37mEndValue:{self.end\_value\_node}, ' \  
 f'(\033[37mStep:{self.step\_value\_node}, (\033[37mBodyFor:{self.body\_node}\033[37m)\033[37m)\033[37m)' \  
 f'\033[37m)\033[37m)\033[37m)'  
  
  
class WhileNode:  
 def \_\_init\_\_(self, condition\_node, body\_node, should\_return\_null):  
 self.condition\_node = condition\_node  
 self.body\_node = body\_node  
 self.should\_return\_null = should\_return\_null  
  
 self.start\_position = self.condition\_node.start\_position  
 self.end\_position = self.body\_node.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[37mWhileNode:(\033[37mConditionWhile:{self.condition\_node}, ' \  
 f'(\033[37mBodyWhile:{self.body\_node}\033[37m)\033[37m)\033[37m)'  
  
  
class ContinueNode:  
 def \_\_init\_\_(self, start\_position, end\_position):  
 self.start\_position = start\_position  
 self.end\_position = end\_position  
  
  
class BreakNode:  
 def \_\_init\_\_(self, start\_position, end\_position):  
 self.start\_position = start\_position  
 self.end\_position = end\_position  
  
  
class FuncDefinitionNode:  
 def \_\_init\_\_(self, var\_name\_token, arguments\_name\_tokens, body\_node, should\_auto\_return):  
 self.var\_name\_token = var\_name\_token  
 self.arguments\_name\_tokens = arguments\_name\_tokens  
 self.body\_node = body\_node  
 self.should\_auto\_return = should\_auto\_return  
  
 if self.var\_name\_token:  
 self.start\_position = self.var\_name\_token.start\_position  
 elif len(self.arguments\_name\_tokens) > 0:  
 self.start\_position = self.arguments\_name\_tokens[0].start\_position  
 else:  
 self.start\_position = self.body\_node.start\_position  
  
 self.end\_position = self.body\_node.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[0mFuncDefNode:(\033[0mVarName:{self.var\_name\_token}, ' \  
 f'(\033[0mArgNameFunc:{self.arguments\_name\_tokens}, ' \  
 f'(\033[0mBody:{self.body\_node}\033[0m)\033[0m)\033[0m)\033[0m)'  
  
  
class CallNode:  
 def \_\_init\_\_(self, node\_to\_call, arguments\_nodes):  
 self.node\_to\_call = node\_to\_call  
 self.arguments\_nodes = arguments\_nodes  
  
 self.start\_position = self.node\_to\_call.start\_position  
  
 if len(self.arguments\_nodes) > 0:  
 self.end\_position = self.arguments\_nodes[len(self.arguments\_nodes) - 1].end\_position  
 else:  
 self.end\_position = self.node\_to\_call.end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[32mCallNode:(\033[32mNodeToCall:{self.node\_to\_call}, ' \  
 f'(\033[32mArgNodeToCall:{self.arguments\_nodes}\033[32m)\033[32m)\033[32m)'  
  
  
class ReturnNode:  
 def \_\_init\_\_(self, node\_to\_return, start\_position, end\_position):  
 self.node\_to\_return = node\_to\_return  
  
 self.start\_position = start\_position  
 self.end\_position = end\_position  
  
 def \_\_repr\_\_(self):  
 return f'(\033[32mReturnNode:(\033[32mReturn:{self.node\_to\_return}\033[32m)\033[32m)'

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Классы для обнаружения ошибок**

from strings\_with\_error import arrow\_string  
  
  
class Error:  
 def \_\_init\_\_(self, start\_position, end\_position, error\_name, details):  
 self.start\_position = start\_position  
 self.end\_position = end\_position  
 self.error\_name = error\_name  
 self.details = details  
  
 def string\_representation(self):  
 result = f'{self.error\_name}: {self.details}\n'  
 result += f'File {self.start\_position.function}, line {self.start\_position.line + 1}'  
 result += '\n\n' + arrow\_string(self.start\_position.function\_text, self.start\_position, self.end\_position)  
 return result  
  
  
class IllegalCharError(Error):  
 def \_\_init\_\_(self, start\_position, end\_position, details):  
 super().\_\_init\_\_(start\_position, end\_position, 'Illegal Character', details)  
  
  
class ExpectedCharError(Error):  
 def \_\_init\_\_(self, start\_position, end\_position, details):  
 super().\_\_init\_\_(start\_position, end\_position, 'Expected Character', details)  
  
  
class InvalidSyntaxError(Error):  
 def \_\_init\_\_(self, start\_position, end\_position, details=''):  
 super().\_\_init\_\_(start\_position, end\_position, 'Invalid Syntax', details)