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

Семантический анализ

Выполнил

Студент гр. 053501

Хиль В.М.

Проверил

Ассистент кафедры информатики

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc132145294)

[2 Краткие теоретические сведения 4](#_Toc132145295)

[3 Примеры работы парсера 5](#_Toc132145296)

[Приложение А (обязательное) Классы для обнаружения ошибок 7](#_Toc132145297)

**1 ЦЕЛЬ РАБОТЫ**

В данной лабораторной работе необходимо показать скриншоты нахождения двух семантических ошибок.

1. **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Семантический анализ является одним из основных этапов теории трансляции. Он представляет собой процесс проверки исходного кода на наличие семантических ошибок, которые не могут быть обнаружены на уровне лексического и синтаксического анализа.

В процессе семантического анализа проводится проверка согласованности типов данных и выражений, а также наличия и корректности использования переменных и функций в программном коде. Этот этап теории трансляции также может включать проверку правильности использования констант, операторов и выражений.

Одной из ключевых задач семантического анализа является поиск ошибок, связанных с неправильным использованием имен и идентификаторов. Например, такие ошибки могут возникать, если переменная объявлена, но не инициализирована, или если в программе используются имена переменных, которые не были определены в предыдущих частях кода.

В случае обнаружения семантических ошибок, транслятор может либо прервать процесс трансляции и выдать сообщение об ошибке, либо продолжить трансляцию, но сигнализировать об ошибке в скомпилированном коде.

В целом, семантический анализ играет важную роль в теории трансляции, так как он позволяет выявлять и исправлять ошибки на ранних этапах разработки программного обеспечения, что в свою очередь улучшает качество и надежность конечного продукта.

1. **ПРИМЕРЫ РАБОТЫ ПАРСЕРА**

Рассмотрим следующую программу, для которой было построено  
синтаксическое дерево в предыдущей лабораторной (см. рисунок 1).

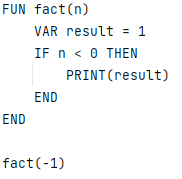


Рисунок 1 – Пример программы

Если добавим, например, в выражение переменную, которая не определена (VAR result = l + k), в этом случае парсер отловит семантическую ошибку (см. рисунок 2).

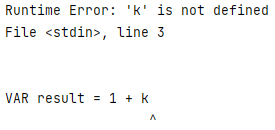


Рисунок 2 – Пример вывода ошибки с неопределённой переменной

Если добавим, например, определение новой переменной k (VAR k = “a”) и попробуем изменить значение переменной result (VAR result = 1 + k), то парсер отловит семантическую ошибку несовместимости типов (см. рисунок 3).

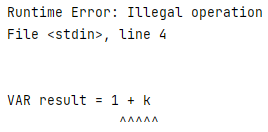


Рисунок 3 – Пример ошибки с несовместимыми типами

Если добавим, например, дополнительный аргумент при вызове функции (fact (-1, 3)), то парсер отловит семантическую ошибку (см. рисунок 4).

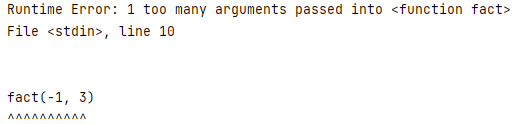


Рисунок 4 – Пример ошибки с передачей аргументов в функцию

**ПРИЛОЖЕНИЕ А**

**(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)  
  
  
class RTError(Error):  
 def \_\_init\_\_(self, start\_position, end\_position, details, context):  
 super().\_\_init\_\_(start\_position, end\_position, 'Runtime Error', details)  
 self.context = context  
  
 def string\_repr(self):  
 result = self.generate\_traceback()  
 result += f'{self.error\_name}: {self.details}'  
 result += '\n\n' + arrow\_string(self.start\_position.function\_text, self.start\_position, self.end\_position)  
 return result  
  
 def generate\_traceback(self):  
 result = ''  
 position = self.start\_position  
 ctx = self.context  
  
 while ctx:  
 result = f' File {position.function}, line {str(position.line + 1)}, in {ctx.display\_name}\n' + result  
 position = ctx.parent\_entry\_pos  
 ctx = ctx.parent  
  
 return 'Traceback (most recent call last):\n' + result