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
2 Краткие теоретические сведения 4
3 Примеры работы парсера 5
Приложение А (обязательное) Классы для обнаружения ошибок 7

1 ЦЕЛЬ РАБОТЫ

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

2 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.

3 ПРИМЕРЫ РАБОТЫ ПАРСЕРА

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

```
FUN fact(n)
    VAR result = 1
    IF n < 0 THEN
        PRINT(result)
    END
END

fact(-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).

```
Runtime Error: 'k' is not defined
File <stdin>, line 3


VAR result = 1 + k
                 ^
```

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).

```
Runtime Error: Illegal operation
File <stdin>, line 4


VAR result = 1 + k
             ^^^^^
```

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).

```
Runtime Error: 1 too many arguments passed into <function fact>
File <stdin>, line 10


fact(-1, 3)
^^^^^^^^^^
```

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
```