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 анализатор

Выполнил

Студент гр. 053501

Волковский О.А.

Проверил

Ассистент кафедры информатики

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1. Цель работы 3](#_Toc128743288)

[2. Краткие теоретические сведения 4](#_Toc128743289)

[3. Примеры работы парсера 5](#_Toc128743290)

[4. Выводы 10](#_Toc128743291)

1. **ЦЕЛЬ РАБОТЫ**

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.).

1. **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Фаза синтаксического анализа и построения синтаксического дерева является важным этапом в теории трансляции. Ее задача - проверить синтаксическую правильность входного языка и создать структуру данных, которая представляет его в виде дерева.

Синтаксический анализатор применяет заданную грамматику и создает синтаксическое дерево, проходя по входному языку. Структура дерева отображает иерархию синтаксических конструкций и их подструктур входного языка. Это дает возможность использовать дерево для дальнейшей обработки, такой как оптимизация и генерация исполняемого кода.

Несмотря на то, что существуют и другие методы анализа входного языка, синтаксический анализ и построение синтаксического дерева являются важными этапами в теории трансляции, позволяя проверить синтаксическую корректность и создать структуру данных для дальнейшей обработки. Синтаксическое дерево является ключевым инструментом для анализа и оптимизации кода.

1. **ПРИМЕРЫ РАБОТЫ ПАРСЕРА**

Рассмотрим следующую программу и построенное на её основе синтаксическое дерево:

int main()

{

int n = 10;

int answer = n \* n;

cout << "n \* n = " << answer << endl;

}

Синтаксическое дерево выглядит следующим образом (см. рисунок 1):

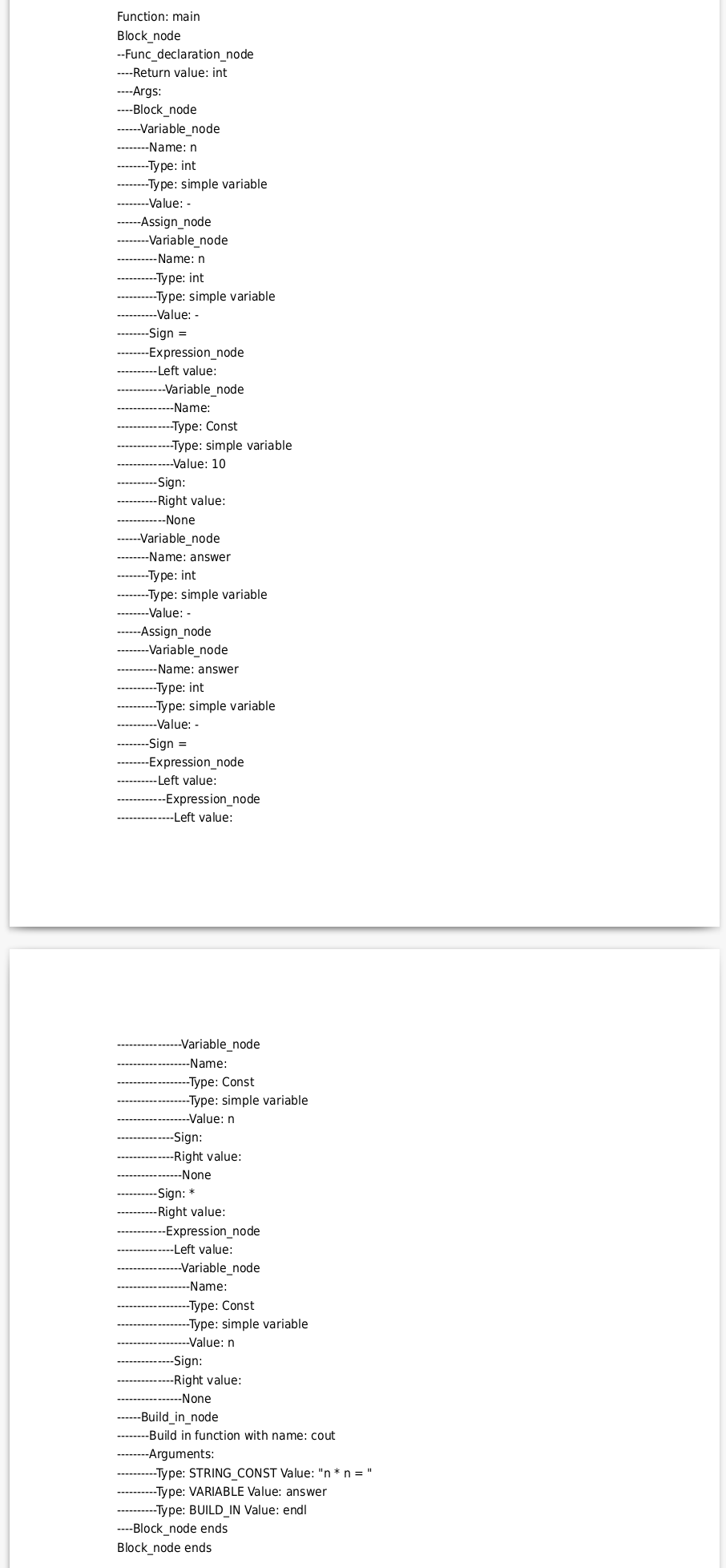


Рисунок 1 – Построенное синтаксическое дерево

Корнем дерева является узел «Block\_node». В нём содержится список утверждений. Утверждение – некоторая логически выполнимая единица. Существуют как специализированные утверждения, например, для ветвлений, циклов, так и утверждения, просто содержащие в себе выражения. Выражение – логическая единица языка, возвращающая некоторое значение при выполнении. Однако в ходе работы парсера необходимо обрабатывать синтаксические ошибки. Добавим, например, второе определение переменной n, в этом случае парсер отловит ошибку (см. рисунок 2).

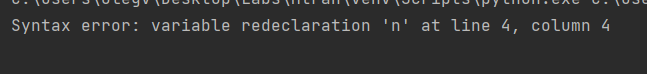


Рисунок 2 – Пример вывода ошибки со вторым определением переменной

Добавим, например, определение переменной, но без имени (double = 0.12) в этом случае парсер отловит ошибку (см. рисунок 3).



Рисунок 3 – Пример ошибки с некорректным определением переменной

Если строка не будет заканчиваться точкой с запятой (int n = 10), то парсер отловит ошибку (см. рисунок 4).



Рисунок 4 – Пример забытой точки с запятой

Обернем тело функции в примере в блок else. В этом случае парсер выдаст ошибку, т.к. нет предшествующего блока if (см. рисунок 5).



Рисунок 5 – Пример восстановления путём создания несуществующих узлов

Также парсер будет отлавливать ошибки при построении некорректных конструкций таких как: циклы, различные вычисления, сравнения, присвоения и др.

1. **ВЫВОДЫ**

Таким образом, в ходе лабораторной работы был реализован парсер подмножества языка С++, который позволяет анализировать написанный на этом языке код.

Было произведено исследование синтаксиса языка С++, чтобы определить основные конструкции и операторы, которые необходимо реализовать в парсере. В результате был разработан алгоритм, основанный на методе рекурсивного спуска, который позволяет обрабатывать и преобразовывать синтаксические конструкции языка в структуры данных.

Реализация парсера включала в себя создание синтаксического анализатора, который использует токены для построения дерева разбора. В процессе работы над парсером было выявлено множество ошибок и неточностей, которые требовали доработки и исправления кода.

Было проведено тестирование парсера на наборе тестовых данных, включающих различные конструкции языка С++. Результаты тестирования показали, что парсер корректно обрабатывает входные данные и выводит ожидаемые результаты.