Национальный исследовательский университет ИТМО
Мегафакультет компьютерных технологий и управления
Факультет программной инженерии и компьютерной техники

Отчет по практическому заданию №1

по курсу «Языки программирования»

Вариант: 1

Руководитель: **Кореньков Юрий Дмитриевич**

ФИО: **Мохаммад Кусае Альосман**

Выполнил студент группы: **[P4119]**

Санкт-Петербург, 2025

Оглавление

1. Цель работы

Целью работы является выбор средства синтаксического анализа и реализация модуля разбора исходного текста в соответствии с языком по варианту 1, построение синтаксического дерева и вывод результата в файл, поддерживающий графическое представление. Язык включает поддержку функций, управляющих конструкций, различных типов данных и операций с ними.

2. План работы

1. Изучить средство синтаксического анализа (ANTLR4) и подготовить спецификацию грамматики в формате Lang.g4.
2. Реализовать модуль разбора, который принимает исходный текст и возвращает синтаксическое дерево (AST) с коллекцией сообщений об ошибках.
3. Реализовать тестовую программу на C++, принимающую имена входного и выходных файлов через аргументы командной строки, выводящую ошибки в stderr и результаты в два файла (текстовый и DOT-формат).
4. Провести тестирование на корректных примерах, примерах с вызовами функций и индексированием, а также на примерах с синтаксическими ошибками и зафиксировать результаты.

3. Описание грамматики языка

Грамматика языка варианта 1 определена в файле Lang.g4 в формате ANTLR4. Она описывает синтаксис для создания программ, включающих функции, переменные, различные выражения и управляющие конструкции. Ниже представлены основные правила грамматики:

source — Корневое правило; определяет программу как набор определений функций.

funcDef — Определение функции, состоящее из сигнатуры и блока кода.

funcSignature — Сигнатура функции: тип возвращаемого значения, имя и список параметров.

block — Блок кода, содержащий объявления переменных и выражения.

statement — Выражение (statement) — базовая единица кода, может быть вызовом функции, присваиванием, условием и т.д.

varDecl — Объявление переменной с типом и инициализацией.

expr — Выражение: может быть функцией call, индексированием, бинарной операцией, идентификатором или литералом.

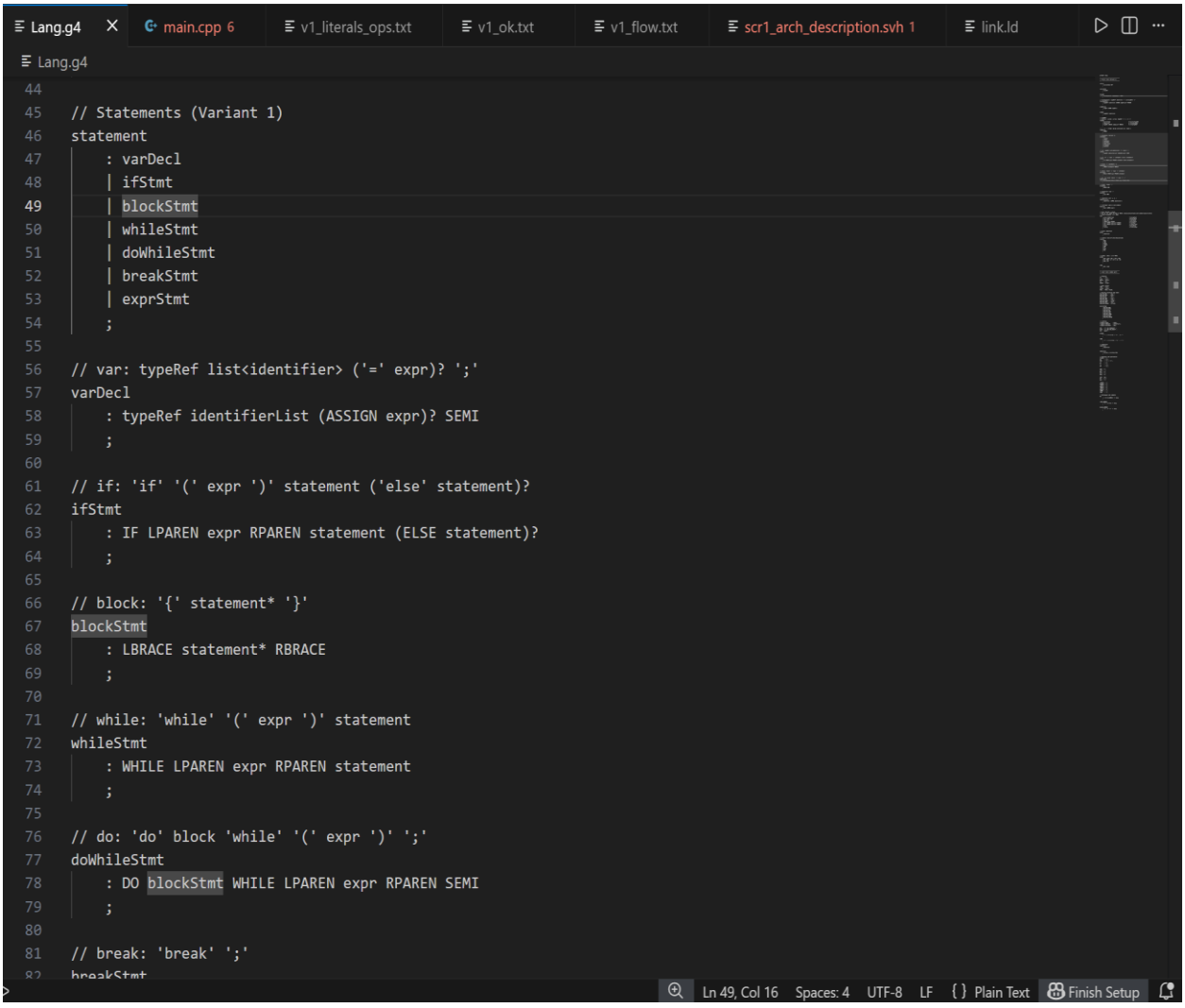
call — Вызов функции: имя функции и список аргументов в скобках.

indexer — Индексирование: выражение в квадратных скобках для доступа к элементу массива.

exprList — Список выражений, разделённых запятыми (параметры функции или элементы).

Полная спецификация грамматики находится в файле Lang.g4. Основная характеристика грамматики варианта 1 — поддержка вложенных вызовов функций, индексирования выражений, условных операторов (if/else), циклов (while, do-while) и различных бинарных операций.

.



4. Ход работы

4.1. Разработка синтаксического анализатора

В качестве средства синтаксического анализа была выбрана библиотека ANTLR4 для генерации лексера и парсера на C++. ANTLR4 позволяет описать грамматику в высокоуровневом формате (Lang.g4) и автоматически генерирует код лексера (LangLexer) и парсера (LangParser), что значительно упрощает разработку.

Грамматика описана согласно синтаксической модели варианта 1, поддерживающей подпрограммы, типы, управляющие конструкции (if/else, while, do-while), выражения, литералы, вызовы функций и индексирование массивов.

4.2. Реализация модуля разбора

Был реализован модуль разбора (ParserModule), использующий сгенерированные ANTLR4 lexer/parser для построения дерева разбора. Модуль принимает исходный текст программы и возвращает структуру с корнем дерева разбора и коллекцией сообщений об ошибках.

Сбор сообщений об ошибках выполняется через собственный обработчик ошибок (ErrorCollector), который перехватывает синтаксические ошибки на этапе лексического и синтаксического анализа. Все ошибки сохраняются и выводятся в stderr в конце обработки.

```cpp
// Чтение входного файла
std::ifstream stream(inputFile);
if (!stream.is_open()) {
    std::cerr << "Ошибка: не удалось открыть файл " << inputFile << "\n";
    return 1;
}

ANTLRInputStream input(stream);
LangLexer lexer(&input);
CommonTokenStream tokens(&lexer);
LangParser parser(&tokens);

// Подключаем сборщик ошибок
ErrorCollector errorCollector;
parser.removeErrorListeners();
parser.addErrorListener(&errorCollector);

// Запуск парсера
LangParser::SourceContext *tree = parser.source();
```

4.3. Дополнительная обработка результата (Parse Tree → AST)

ANTLR4 формирует Parse Tree (дерево разбора), которое отражает структуру грамматики с максимальной точностью, включая все терминальные и нетерминальные символы. Однако для удобства анализа и визуализации была выполнена дополнительная обработка: построение собственного AST (Abstract Syntax Tree), где каждый узел соответствует значимой синтаксической конструкции языка.

В AST представлены узлы следующих типов: funcDef (определение функции), funcSignature (сигнатура), block (блок кода), varDecl (объявление переменной), exprStmt (выражение-оператор), if/while/doWhile (управляющие конструкции), call (вызов функции), indexer (индексирование), binaryOp (бинарная операция), unaryOp (унарная операция), literalValue (литерал) и другие.

Данная обработка реализована в модуле ASTBuilder, который обходит дерево ANTLR и формирует иерархию узлов ASTNode. Каждый узел содержит имя (тип конструкции), значение (для литералов и идентификаторов) и список дочерних узлов.

4.4

Тес
зап

src > visitor > C ASTBuilder.h > ASTBuilder > visitDoWhileStmt(LangParser::DoWhileStmtContext *)

```cpp
#pragma once

#include <any>
#include "../../generated/LangBaseVisitor.h"

class ASTBuilder : public LangBaseVisitor {
public:
    //ASTBuilder.cpp التي نستخدمها في overrides نكتفي بالـ
    std::any visitSource(LangParser::SourceContext *ctx) override;
    std::any visitFuncDef(LangParser::FuncDefContext *ctx) override;
    std::any visitFuncSignature(LangParser::FuncSignatureContext *ctx) override;
    std::any visitArgDefList(LangParser::ArgDefListContext *ctx) override;
    std::any visitArgDef(LangParser::ArgDefContext *ctx) override;

    std::any visitBlockStmt(LangParser::BlockStmtContext *ctx) override;
    std::any visitVarDecl(LangParser::VarDeclContext *ctx) override;
    std::any visitIfStmt(LangParser::IfStmtContext *ctx) override;
    std::any visitWhileStmt(LangParser::WhileStmtContext *ctx) override;
    std::any visitDoWhileStmt(LangParser::DoWhileStmtContext *ctx) override;
    std::any visitBreakStmt(LangParser::BreakStmtContext *ctx) override;
    std::any visitExprStmt(LangParser::ExprStmtContext *ctx) override;

    std::any visitBuiltinTypeRef(LangParser::BuiltinTypeRefContext *ctx) override;
    std::any visitCustomTypeRef(LangParser::CustomTypeRefContext *ctx) override;
    std::any visitArrayTypeRef(LangParser::ArrayTypeRefContext *ctx) override;
    std::any visitCommaList(LangParser::CommaListContext *ctx) override;
```

файла (исходный код) и имена двух выходных файлов (текстовое представление AST и графическое представление в формате Graphviz DOT).

При обнаружении синтаксических ошибок все сообщения об ошибках выводятся в стандартный поток ошибок (stderr), что позволяет пользователю видеть описание проблем при запуске. Если парсинг успешен, выводится сообщение об успешном завершении.

Выходной файл в формате DOT может быть легко преобразован в изображение (PNG, PDF и т.д.) с помощью утилиты Graphviz:

```
./build/ProgLangLab1 <input.txt> <output.txt> <output.dot>
dot -Tpng <output.dot> -o <output.png>
```

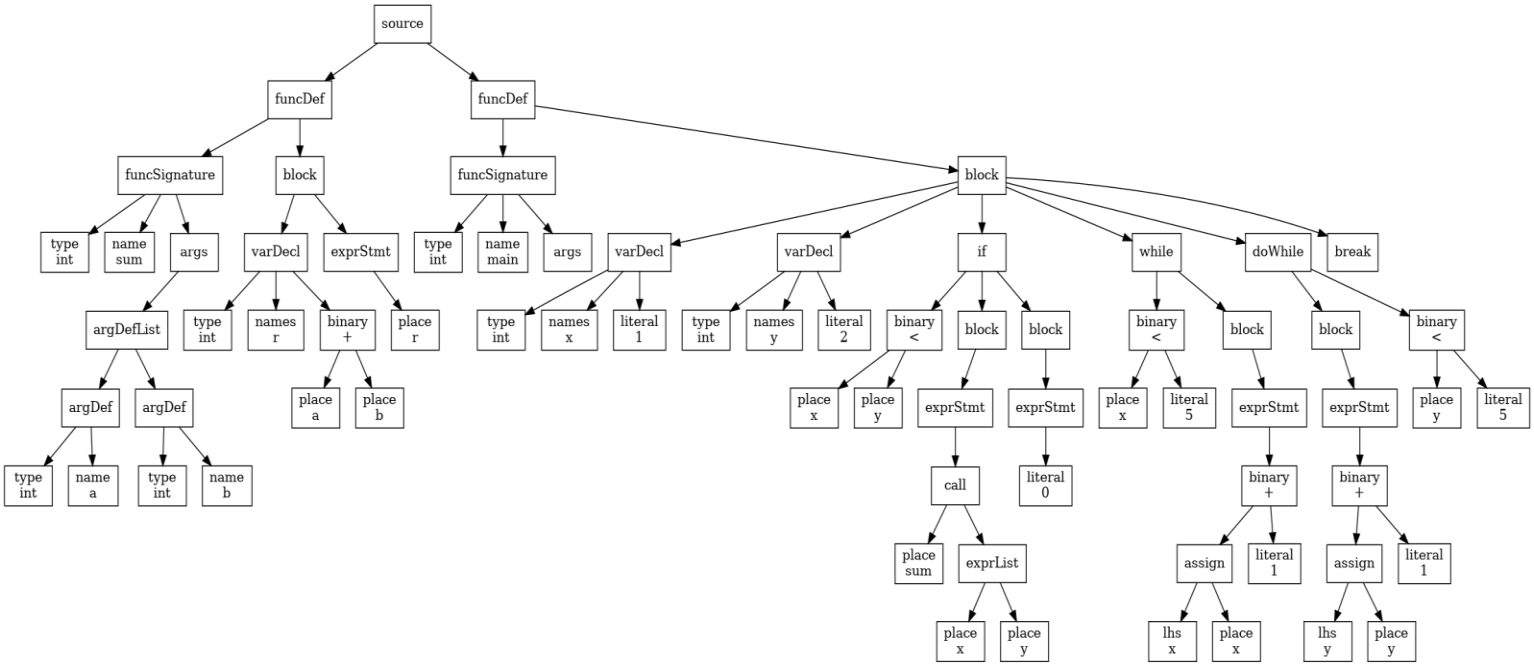
```
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$ cmake -S . -B build
-- Configuring done (0.1s)
-- Generating done (0.2s)
-- Build files have been written to: /mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main/build
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$ cmake --build build -j 8
[100%] Built target ProgLangLab1
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$ ./build/ProgLangLab1 ex1.lang out
_ast.txt
Использование: ./build/ProgLangLab1 <входной_файл> <выходной_txt> <выходной_dot>
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$ cat out_ast.txt
cat: out_ast.txt: No such file or directory
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$ ./build/ProgLangLab1 ex1.lang out
_ast.txt out_ast.dot
Парсинг завершен без ошибок.
AST (txt) записан в файл: out_ast.txt
AST (dot) записан в файл: out_ast.dot
```

5. Тестирование

5.1. Корректный пример (управляющие конструкции)

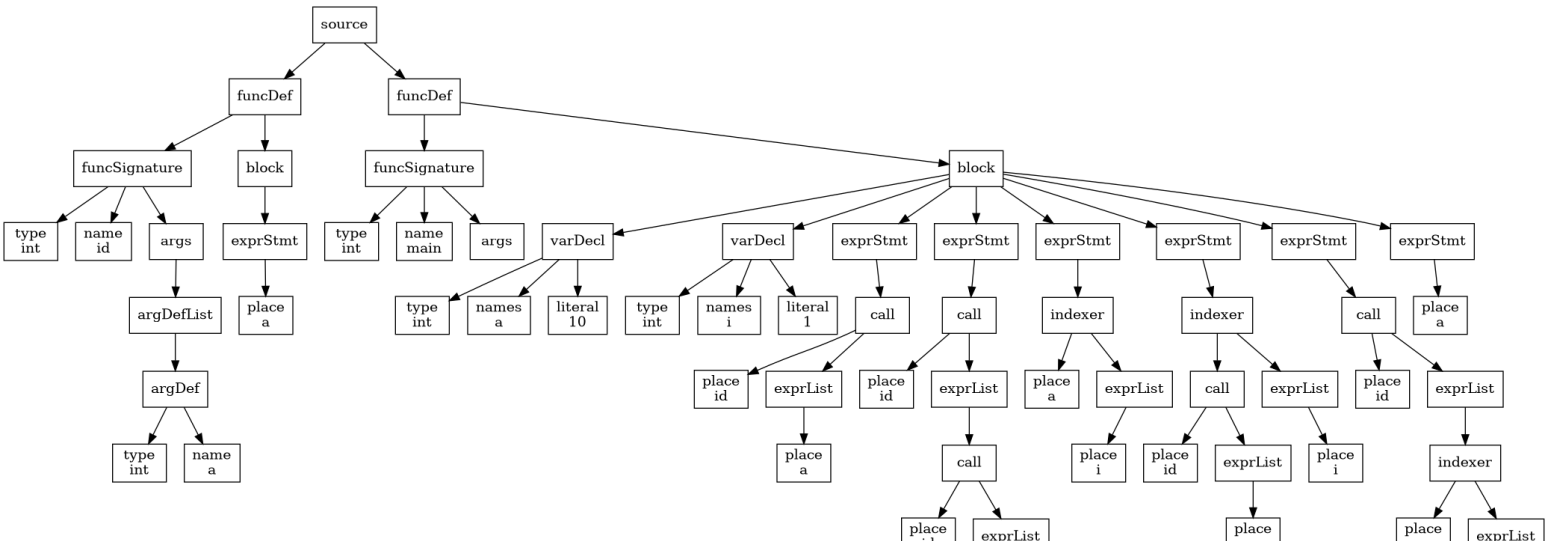
В качестве первого теста был использован файл tests/v1_ok.txt, содержащий объявления функций, условные операторы (if/else), циклы (while и do-while), объявления переменных и различные выражения.

Этот тест демонстрирует, что парсер корректно обрабатывает базовые конструкции языка: определение функций с возвращаемым типом и параметрами, управляющие структуры, управление потоком выполнения и правильное формирование синтаксического дерева. Успешное прохождение этого теста показывает, что основной функционал парсера работает как ожидается.



5.2. Пример с вызовами и индексированием

Второй тест — файл tests/v1_calls_indexer.txt — демонстрирует вызовы функций, вложенные вызовы функций и индексирование выражений. Примеры включают простые вызовы (id(a)), вложенные вызовы (id(id(a))), индексирование (a[i]), индексирование результатов вызовов (id(a)[i]) и индексирование параметров функций (id(a[i])).



5.3. Пример с ошибками

Третий тест — файл tests/v1_errors.txt — содержит намеренные синтаксические ошибки для проверки корректности обработки ошибок парсером. Примеры ошибок включают: пропущенное выражение после оператора присваивания (=), пропущенные точки с запятой (;) в конце выражений, неправильную структуру конструкций.

При запуске тестовой программы на файле с ошибками все синтаксические ошибки должны быть обнаружены, описаны (с указанием строки и типа ошибки) и выведены в stderr. Это соответствует требованиям задания и демонстрирует, что парсер обладает хорошей системой обработки и сообщения об ошибках.

```
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$ cat tests/v1_errors.txt
int main() {
  int x = ;
  if (x < 3) {
    x = x + 1
  } else {
    x = x + 2;
  }
}
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$ ./build/ProgLangLab1 tests/v1_err
ors.txt out_err.txt out_err.dot
Ошибки парсера:
  PARSER error at 2:10 - mismatched input ';' expecting {'true', 'false', BOOL, HEX, BITS, DEC, STRING, CHAR, Identifier
, '-', '!', '('} (offending: ";")
  PARSER error at 5:2 - missing ';' at '}' (offending: "}")
qusae@WIN-L2A48AM4C55:/mnt/c/Users/qusae/OneDrive/Desktop/Master/C/ProgLang_lab1-main$
```

6. Обработка ошибок

Система обработки ошибок реализована на двух уровнях:

Уровень лексического анализа: Ошибки на этом уровне возникают при обнаружении неправильных или неизвестных символов. Лексер (LangLexer) отказывает в обработке и формирует сообщение об ошибке с указанием позиции в исходном коде.

Уровень синтаксического анализа: Ошибки на этом уровне возникают, когда последовательность токенов не соответствует правилам грамматики. Парсер (LangParser) останавливается на неправильной конструкции, но при использовании стратегии ошибок ANTLR4 может попытаться восстановиться и продолжить анализ для обнаружения дополнительных ошибок.

Все собранные ошибки хранятся в коллекции ErrorCollector, которая включает информацию о типе ошибки, строке кода, позиции символа и описание проблемы. После завершения парсинга все ошибки выводятся в stderr в читаемом формате, что позволяет разработчику быстро найти и исправить проблемы в исходном коде.

Примеры тестов с ошибками показывают, что парсер:

- Обнаруживает синтаксические ошибки на разных позициях в коде
- Сообщает точное положение ошибки (строка и столбец)
- Предоставляет понятное описание того, что было ожидается
- Продолжает анализ для обнаружения возможных дополнительных ошибок
- Выводит все ошибки в stderr как требуется в задании

7. Заключение

В ходе работы был успешно реализован модуль синтаксического анализа для языка программирования варианта 1. Модуль основан на ANTLR4 и обеспечивает полный цикл разбора исходного кода: лексический анализ, синтаксический анализ, построение синтаксического дерева (AST) и формирование сообщений об ошибках.

Ключевые достижения:

5. Разработана полная грамматика языка в формате ANTLR4 (Lang.g4), поддерживающая все требуемые конструкции
6. Реализована система построения AST из дерева разбора ANTLR4 с правильной иерархией узлов
7. Создана тестовая программа на C++, принимающая входные/выходные файлы через аргументы командной строки
8. Реализована система обработки ошибок с выводом в stderr и подробными сообщениями о проблемах
9. Проведено тестирование на примерах с управляющими конструкциями, вложенными вызовами функций, индексированием и синтаксическими ошибками
10. Результаты представлены как в текстовом формате, так и в виде графического представления (Graphviz DOT)

Реализованный парсер готов к использованию и может успешно обрабатывать исходный код на языке варианта 1, выявляя синтаксические ошибки и формируя удобное для анализа представление синтаксического дерева.