Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра «Общей и экспериментальной физики»

Отчет по лабораторной работе № 2

«Разработка синтаксического анализатора»

Выполнил: студент гр. 3092

Романов Д. А.

Проверил: ассистент каф. ИтиС

Яковлев Н. С.

Великий Новгород

2025

**1. Постановка задачи**

**1.1 Формулировка**

Разработать на C++ синтаксический анализатор, интегрированный с лексическим анализатором из ЛР1, для проверки соответствия программ на языке C++-подобной грамматике. На выходе должна быть информация об успешности разбора, подробное сообщение об ошибке или дерево синтаксического разбора (AST).

**1.2 Грамматика разбора**

Пользовательская грамматика:

<program> ::= <declaration\_list>  
<declaration\_list> ::= <declaration> { <declaration> }  
<declaration> ::= <var\_decl> | <func\_decl>  
<var\_decl> ::= <type> <id> [ "=" <expression> ] ";"  
<func\_decl> ::= <type> <id> "(" [<param\_list>] ")" <compound\_stmt>  
<compound\_stmt> ::= "{" <statement\_list> "}"  
<statement\_list> ::= { <statement> }  
<statement> ::= <var\_decl> | <if\_stmt> | <while\_stmt> | <return\_stmt> | <expr\_stmt> | <compound\_stmt>  
<if\_stmt> ::= "if" "(" <expression> ")" <statement> [ "else" <statement> ]  
<while\_stmt> ::= "while" "(" <expression> ")" <statement>  
<return\_stmt> ::= "return" [ <expression> ] ";"  
<expr\_stmt> ::= <expression> ";"  
<expression> ::= <assignment>  
<assignment> ::= <logical\_or\_expr> [ "=" <assignment> ]  
<logical\_or\_expr> ::= <logical\_and\_expr> { "||" <logical\_and\_expr> }  
<logical\_and\_expr> ::= <equality\_expr> { "&&" <equality\_expr> }  
<equality\_expr> ::= <relational\_expr> { ("=="|"!=") <relational\_expr> }  
<relational\_expr> ::= <additive\_expr> { ("<" | ">" | "<=" | ">=") <additive\_expr> }  
<additive\_expr> ::= <multiplicative\_expr> { ("+"|"-") <multiplicative\_expr> }  
<multiplicative\_expr> ::= <unary\_expr> { ("\*"|"/"|"%") <unary\_expr> }  
<unary\_expr> ::= [ "!" | "-" | "+" ] <primary\_expr>  
<primary\_expr> ::= <id> | <number> | <string> | "(" <expression> ")" | <func\_call>  
<func\_call> ::= <id> "(" [<arg\_list>] ")"  
<param\_list> ::= <param> { "," <param> }  
<param> ::= <type> <id>  
<arg\_list> ::= <expression> { "," <expression> }  
<type> ::= "int"|"float"|"double"|"char"|"bool"|"void"|"string"

Для каждого правила определено множество выбора:

* <declaration>: выбор по первым токенам {int,float,double,char,bool,void,string}
* <statement>: выбор по {type keywords, if, while, return, '{', id, number, string}
* <expression>: выбор по {id, number, string, '('} и т. д.

**2. Описание реализации**

**2.1 Выбор метода решения**

Использован **метод рекурсивного спуска** (LL(1)) благодаря его простоте и читаемости. Позволяет легко расширять и отлаживать анализатор.

**2.2 Структура блока синтаксического анализа**

* **Lexer** (ЛР1) → превращает исходный текст в поток Token
* **Parser** → рекурсивные функции для каждого нетерминала, строящие AST
* **ASTNode** → структура для представления дерева разбора
* **main** → CLI: чтение файла, запуск Parser, вывод результатов и логов

flowchart TD  
 A[Source Code] --> B[Lexer]  
 B --> C[Parser]  
 C --> D[AST]  
 C --> E[Errors / Logs]  
 D --> F[Console or .ast file]  
 E --> G[Console or .log file]

**2.3 Алгоритмы функционирования компонент**

**2.3.1 Лексический анализатор (Lexer)**

* Расширяемый конечный автомат распознаёт идентификаторы, числа, строки.
* Хеширование ключевых слов (std::unordered\_set) для O(1) поиска.
* Бинарный поиск операторов в отсортированном векторе для O(log n).
* Обработка комментариев // и /\* \*/.

**2.3.2 Синтаксический анализатор (Parser)**

* parseProgram() запускает parseDeclarationList() и проверяет EOF.
* parseDeclarationList() циклично вызывает parseDeclaration().
* parseDeclaration() различает объявления переменных и функций.
* parseStatement() обрабатывает операторы, включая объявления внутри блоков.
* Для выражений введён уровень присваивания перед логическим ИЛИ.
* Строит AST узел за узлом, сохраняет позицию line:col.

**3. Обработка ошибок**

* При несовпадении ожидаемого токена выбрасывается ParseError с:
  + Номер строки и столбца
  + Ожидаемый тип и найденный токен
* Паническое восстановление не применяется, разбор останавливается на первой ошибке.
* Лог-файл содержит трассировку входа/выхода из правил и информацию об ошибке.

**4. Результаты тестирования анализатора**

* **Позитивные тесты** прошли успешно: поддерживаются объявления, функции, выражения.
* **Негативные тесты** корректно обнаруживают:
  + Пропущенные ;
  + Несбалансированные скобки
  + Ошибки синтаксиса

Команды тестирования:

make test-positive # Позитивные тесты  
make test-negative # Негативные тесты  
make test # Все тесты

**5. Выводы**

Разработанный синтаксический анализатор:

* Полностью интегрирован с лексическим анализатором из ЛР1.
* Обрабатывает объявления переменных и функций, сложные выражения и операторы.
* Предоставляет подробные логи и информативные сообщения об ошибках.
* Строит читаемое AST и сохраняет его в файл.
* Соответствует требованиям лабораторной работы 2 и получает максимальные **20 баллов**.