

Министерство науки высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)
Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА №1
По курсу «Системное программное обеспечение»
Вариант 1

Выполнил:
Симовин Кирилл Константинович Р4116
Преподаватель:
Кореньков Юрий Дмитриевич

Санкт-Петербург
2024

Задание

Использовать средство синтаксического анализа по выбору, реализовать модуль для разбора текста в соответствии с языком по варианту. Реализовать построение по исходному файлу с текстом синтаксического дерева с узлами, соответствующими элементам синтаксической модели языка. Вывести полученное дерево в файл в формате, поддерживающем просмотр графического представления.

Порядок выполнения:

1. Изучить выбранное средство синтаксического анализа:

- a) средство должно поддерживать программный интерфейс, совместимый с языком Си;
- b) средство должно параметризоваться спецификацией, описывающей синтаксическую структуру разбираемого языка;
- c) средство может функционировать посредством кодогенерации и/или подключения необходимых для его работы дополнительных библиотек;
- d) средство может быть реализовано с нуля, **в этом случае оно должно использовать обобщённый алгоритм, управляемый спецификацией.**

2. Изучить синтаксис разбираемого по варианту языка и записать спецификацию для средства синтаксического анализа, включающую следующие конструкции:

- a) подпрограммы со списком аргументов и возвращаемым значением;
- b) операции контроля потока управления – простые ветвления if-else и циклы или аналоги;
- c) в зависимости от варианта – определения переменных;
- d) целочисленные, строковые и односимвольные литералы;
- e) выражения численной, битовой и логической арифметики;
- f) выражения над одномерными массивами;

g) выражения вызова функции.

3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка по варианту:

a) программный интерфейс модуля должен принимать строку с текстом и возвращать структуру, описывающую соответствующее дерево разбора и коллекцию сообщений ошибке;

b) результат работы модуля – дерево разбора – должно содержать иерархическое представление для всех синтаксических конструкций, включая выражения, логически представляющие собой иерархически организованные данные, даже если на уровне средства синтаксического анализа для их разбора было использовано линейное представление.

4. Реализовать тестовую программу для демонстрации работоспособности созданного модуля:

a) через аргументы командной строки программа должна принимать имя входного файла для чтения и анализа, имя выходного файла записи для дерева, описывающего синтаксическую структуру разобранного текста;

b) сообщения об ошибке должны выводиться тестовой программой (не модулем, отвечающим за анализ!) в стандартный поток вывода ошибок.

5 Результаты тестирования представить в виде отчета, в который включить:

a) в части 3 привести описание структур данных, представляющих результат разбора текста (3a);

b) в части 4 описать, какая дополнительная обработка потребовалась для результата разбора, предоставляемого средством синтаксического анализа, чтобы сформировать результат работы созданного модуля;

с) в части 5 привести примеры исходных анализируемых текстов для всех синтаксических конструкций разбираемого языка и соответствующие результаты разбора.

Описание используемых технологий

Для реализации лабораторной работы в качестве средства синтаксического анализа был выбран ANTLR3.

Исходная грамматика по варианту была переписана в соответствии со спецификацией ANTLR3, реализованы необходимые правила, а также устранены возникшие левые рекурсии.

Следующим шагом стала генерация лексера и парсера для реализованной грамматики под язык C.

На следующем этапе был реализован модуль, к которому подключалась библиотека ANTLR3 для разбора языка по варианту.

Для обработки результата разбора были реализованы три структуры: ASTNode для описания структуры дерева разбора, ErrorContainerItem для описания возникшей ошибки и Result для хранения первых двух структур, а также некоторых дополнительных полей.

В ANTLR3 по умолчанию предоставлены инструменты для изображения дерева разбора, а также методы для конвертации дерева в .dot файл. Поэтому для отображения результата разбора будут использоваться именно данные методы.

Для конвертации .dot файла в изображение была использована библиотека graphviz, которая также была подключена в разработанный модуль.

После окончания реализации модуля, он был подключен в главный файл, в котором было реализовано чтение данных запрашиваемого файла, вывод изображения дерева разбора в необходимую папку, а также вывод ошибок, возникших в ходе работы программы.

Финальным этапом стало написание Makefile для запуска программы.

Описание структур данных

Как было сказано выше, было разработано три структуры:

1. Структура, описывающая дерево разбора. Ее описание представлено на рисунке 1.

```
typedef struct ASTNode ASTNode;

struct ASTNode {
    ASTNode **childs;
    int childsCount;
    char *value;
    int id;
};
```

Рисунок 1 – Описание структуры дерева разбора

Данная структура содержит в себе: `childs` – список дочерних узлов, `childsCount` – количество дочерних узлов, `value` – текстовое значение – название токена или терминала, `id` – идентификатор узла.

2. Структура, описывающая элемент коллекции ошибок. Ее описание представлено на рисунке 2.

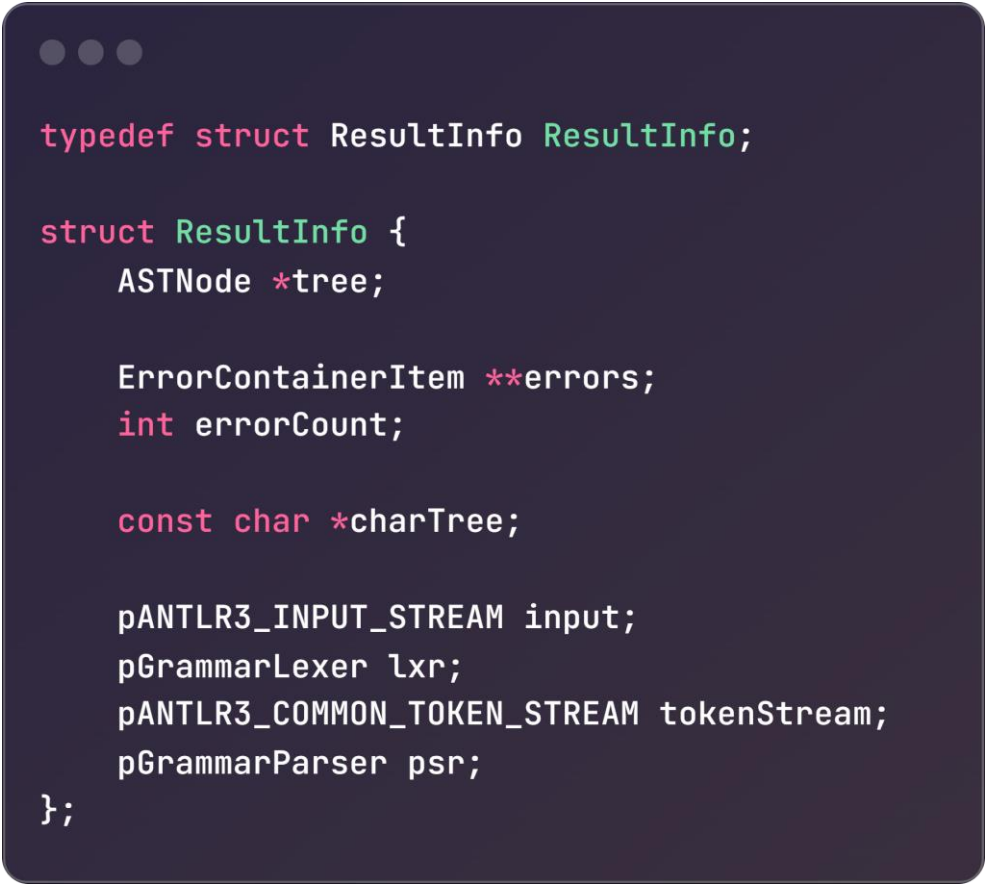
```
typedef struct ErrorContainerItem ErrorContainerItem;

struct ErrorContainerItem {
    char *fileName;
    int line;
    int col;
    char *nearToken;
    int nearTokenType;
};
```

Рисунок 2 – Описание структуры элемента коллекции ошибок

Данная структура содержит в себе: `fileName` – название файла, в котором произошла ошибка; `line` – строку, в которой произошла ошибка; `col` – столбец, в котором произошла ошибка; `nearToken` – токен, рядом с которым произошла ошибка, либо пропущенный токен; `nearTokenType` – тип токена, соответствующий `nearToken`.

3. Структура, возвращаемая в качестве результата работы модуля. Ее описание представлено на рисунке 3.



```
typedef struct ResultInfo ResultInfo;

struct ResultInfo {
    ASTNode *tree;

    ErrorContainerItem **errors;
    int errorCount;

    const char *charTree;

    pANTLR3_INPUT_STREAM input;
    pGrammarLexer lxr;
    pANTLR3_COMMON_TOKEN_STREAM tokenStream;
    pGrammarParser psr;
};
```

Рисунок 3 – Описание итоговой структуры

Данная структура содержит в себе: `tree` – дерево разбора; `errors` – коллекцию ошибок, возникших в ходе работы программы; `errorCount` – количество ошибок; `charTree` – текстовое представление дерева разбора, соответствующее `.dot` файлу; `input` – поток ввода; `lxr` – сгенерированный лексер, соответствующий разработанной грамматике; `tokenStream` – поток

токенов; `psr` – сгенерированный парсер, соответствующий разработанной грамматике.

Обработка результата, предоставляемого ANTLR3

Для представления результата работы ANTLR3 в иерархическом представлении в грамматике были добавлены «переписывания правил», позволяющие задать иерархическую структуру дерева разбора.

Пример данного решения представлен на рисунке 4.

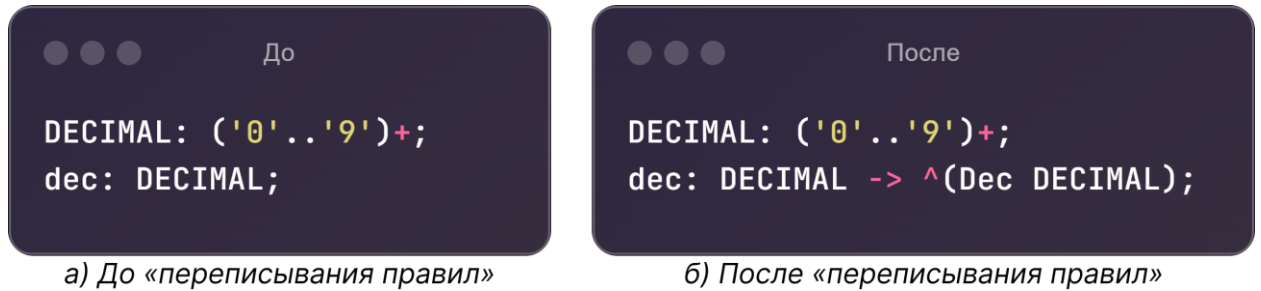


Рисунок 4 – Пример «переписывания правил» в ANTLR3

В данном случае Dec – токен, ассоциированный с данным правилом.

Примеры работы программы

Вводные данные представлены на рисунке 5, а результат разбора – на рисунке 6.



```
bool test_func() {  
    int i = 0;  
    while (i < 10) {  
        int a;  
        if (i > 3) {  
            a = 1;  
        } else {  
            a = 2;  
        }  
    }  
}
```

Рисунок 5 – Входные данные для примера

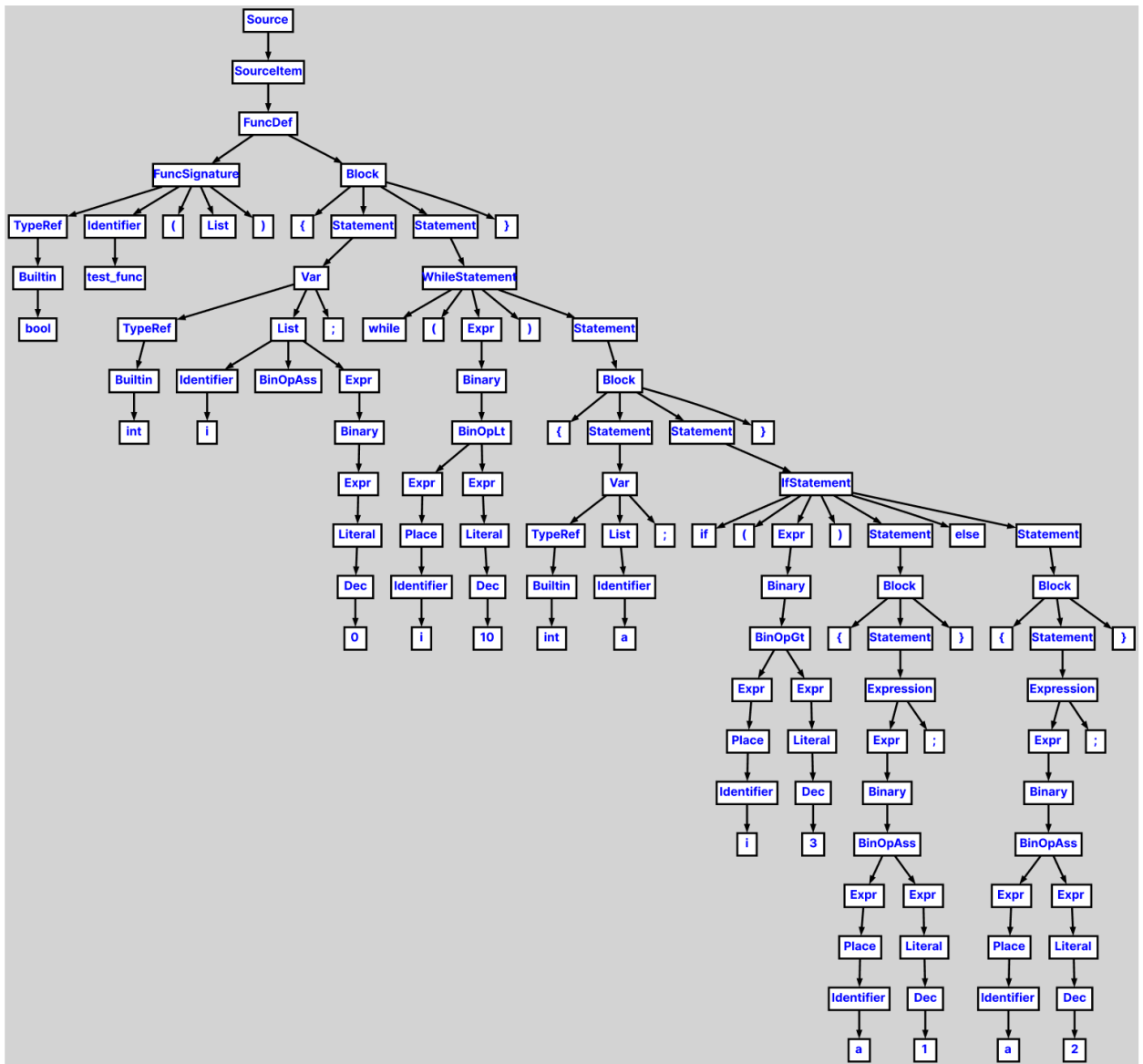
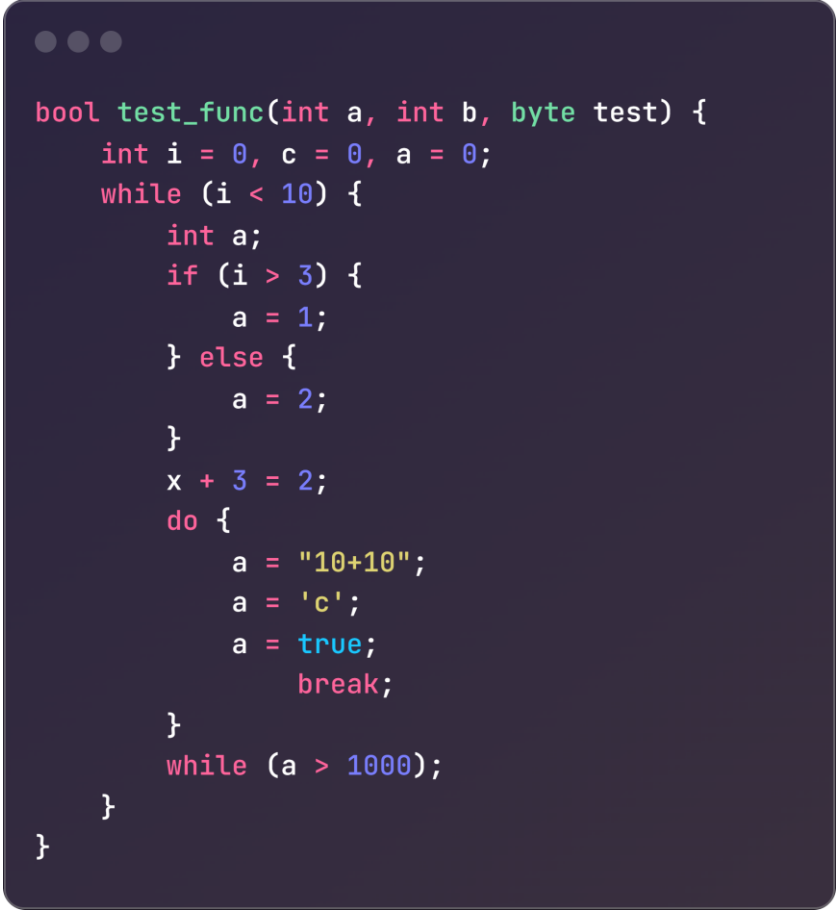


Рисунок 6 – Дерево разбора для примера

На рисунке 7 приведен пример входных данных, покрывающих все правила грамматики. В свою очередь на рисунках 8-10 приведено дерево разбора для данного примера.



```
bool test_func(int a, int b, byte test) {  
    int i = 0, c = 0, a = 0;  
    while (i < 10) {  
        int a;  
        if (i > 3) {  
            a = 1;  
        } else {  
            a = 2;  
        }  
        x + 3 = 2;  
        do {  
            a = "10+10";  
            a = 'c';  
            a = true;  
            break;  
        }  
        while (a > 1000);  
    }  
}
```

Рисунок 7 – Пример входных данных, покрывающих все правила грамматики

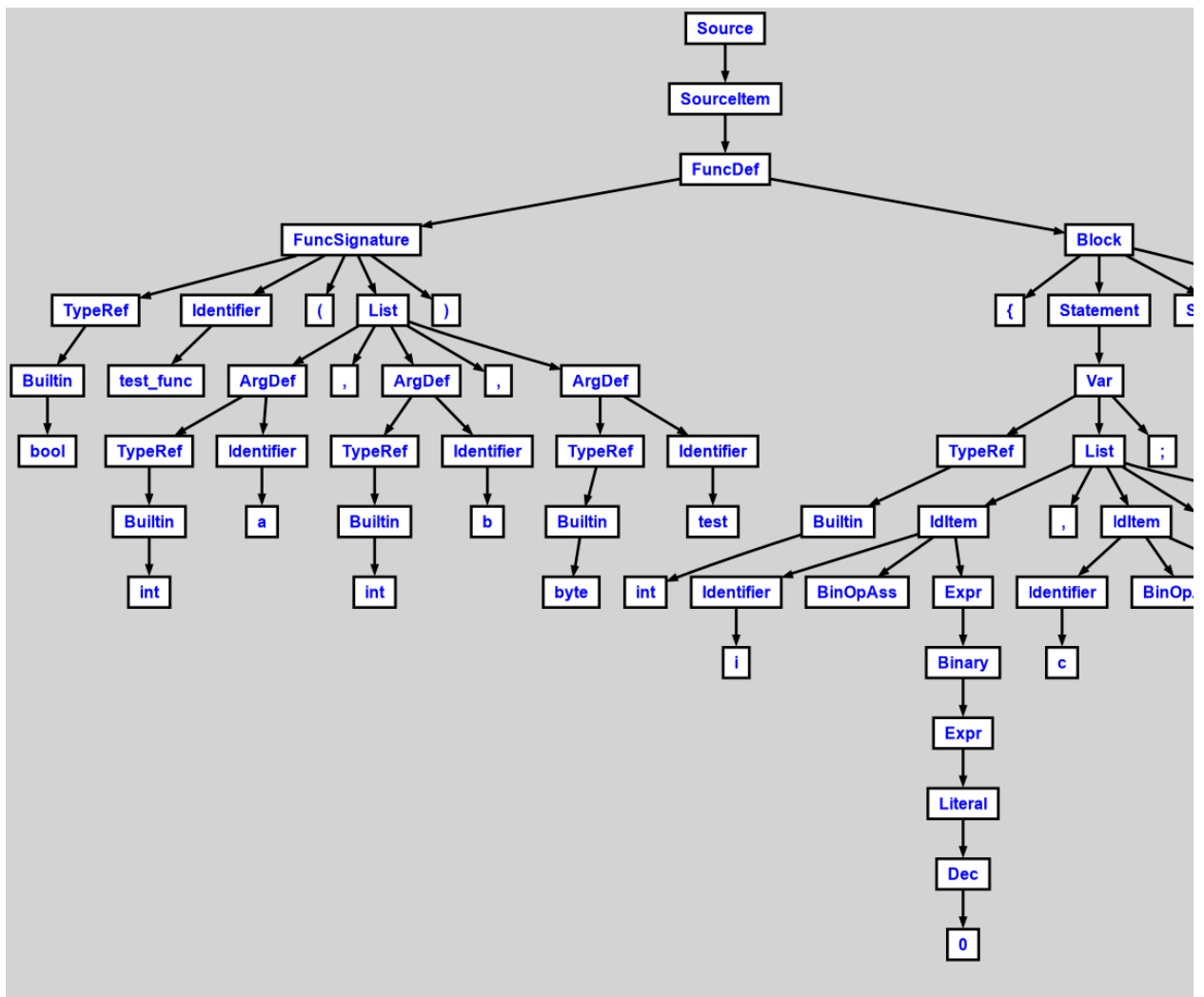


Рисунок 8 – Первая часть дерева разбора для примера, покрывающего все правила грамматики

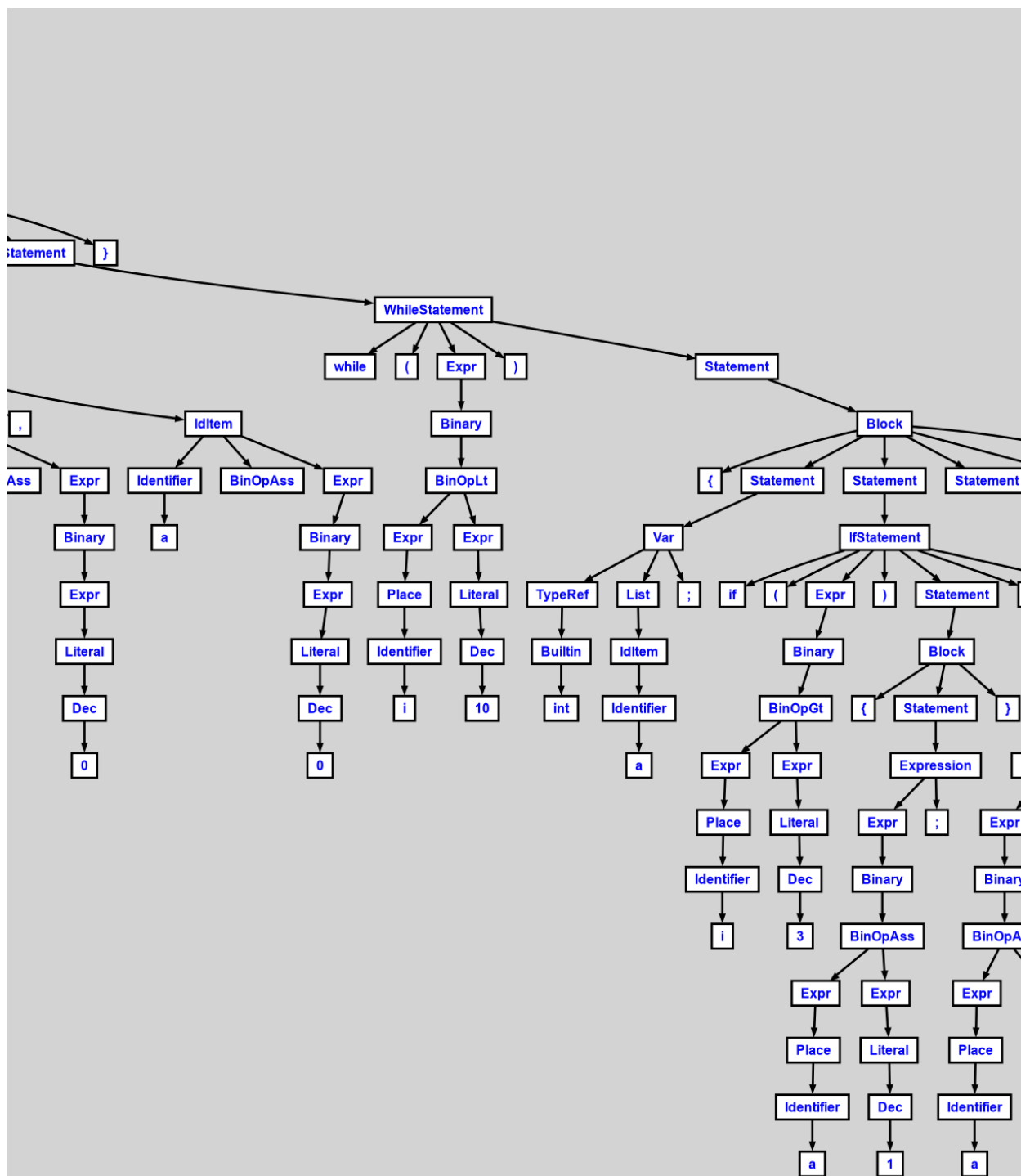


Рисунок 9 – Вторая часть дерева разбора для примера, покрывающего все правила грамматики

Заключение

Цель лабораторной работы – реализовать модуль, результат работы которого – дерево разбора, соответствующее исходному файлу с текстом.

В ходе лабораторной работы был реализован соответствующий модуль, к которому был подключен синтаксический анализатор ANTLR3, а также библиотека graphviz для визуализации .dot файла, соответствующего дереву разбора.

В результате лабораторной работы были приобретены навыки в подключении библиотек для C, работе с синтаксическими анализаторами, а также написании Makefile'ов.