Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Низкоуровневое программирование
Лабораторная работа №2

Выполнил студент: **Соловьев П.А.**

Группа: **P33302**

Преподаватель: **Кореньков Ю.Д.**

Санкт-Петербург
2023

Цель

Реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных.

Задачи

1. Изучить выбранное средство синтаксического анализа.
2. Изучить синтаксис языка запросов и записать спецификацию для средства синтаксического анализа.
3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов.
4. Реализовать тестовую программу для демонстрации работоспособности созданного модуля, принимающую на стандартный ввод текст запроса и выводящую на стандартный вывод результирующее дерево разбора или сообщение об ошибке.

Исходный код проекта

https://github.com/Moleus/llp-lab2

Описание работы

Программа реализована на языке C и представляет собой cli приложение, которое принимает в стандартных вход запрос и выводит его дерево разбора.

Программа состоит из следующих модулей:

- parser.y - описание грамматики языка запросов на языке Bison
- lexer.l - описание лексического анализатора на языке Flex
- main.c - точка входа в программу
- types - описание типов данных, используемых в программе и вспомогательных функций для работы с ними

Примеры запуска программы:

```
→ sem5/llp/lab2 git:(main) × ./main <<< "$(echo -n 'a/b[@x=1]')"
Parsed Query:
node name: a
        node name: b
        Filters for node:
          filter type: select property by condition
          arg name: x
          operation: ==
          arg value: 1
```

Рисунок 1: Вывод элемента, у которого значение атрибута x = 1

```
→ sem5/llp/lab2 git:(main) × ./main <<< "$(echo -n 'create(foo[@bar=42])')"
Parsed Query:
node name: foo
Filters for node:
  filter type: select property by condition
  arg name: bar
  operation: ==
  arg value: 42

Applying function: create
```

Рисунок 2: Добавление корневого элемента foo с атрибутом bar = 42

```
→ sem5/llp/lab2 git:(main) × ./main <<< "$(echo -n 'create(foo/bar[@baz=42])')"
Parsed Query:
node name: foo
        node name: bar
        Filters for node:
          filter type: select property by condition
          arg name: baz
          operation: ==
          arg value: 42

Applying function: create
```

Рисунок 3: Добавление дочернего элемента bar к foo с атрибутом baz = 42

```
→ sem5/llp/lab2 git:(main) × ./main <<< "$(echo -n 'people[@name=Andrey]/age')"
Parsed Query:
node name: people
Filters for node:
  filter type: select property by condition
  arg name: name
  operation: ==
  arg value: Andrey
        node name: age
```

Рисунок 4: Вывод age у элементов people, у которых name = 'Andrey'

```
→ sem5/llp/lab2 git:(main) × ./main <<< "$(echo -n 'delete(foo[@bar=hello]/baz)')"
Parsed Query:
node name: foo
Filters for node:
  filter type: select property by condition
  arg name: bar
  operation: ==
  arg value: hello
        node name: baz

Applying function: delete
```

Рисунок 5: Удаление всех дочерних элементов baz, у которых у foo атрибут bar = 'hello'

Аспекты реализации

Для реализации в синтаксис XPath были добавлены указания операций добавления и удаления, чтобы была возможность изменять элементы в документном дереве. Поддерживаемые функции: `create()` - создание элемента, `delete()` - удаление элемента, `update()` - обновление элемента.

Описание структур данных:

Основной структурой является `Element`, она хранит в себе значение и тип элемента, и представляет собой элемент дерева. Тип элемента может быть одним из следующих: `int`, `bool`, `string`, `double`.

```
 7 typedef enum {
 8     BOOLEAN_TYPE = 1,
 9     NUMBER_TYPE,
10     DOUBLE_TYPE,
11     STRING_TYPE
12 } ValueType;
13
14 typedef struct {
15     ValueType type;
16     union {
17         bool boolean;
18         int32_t number;
19         double double_number;
20         char string[MAX_STRING_SIZE];
21     };
22 } Element;
```

Рисунок 6: Структура Element

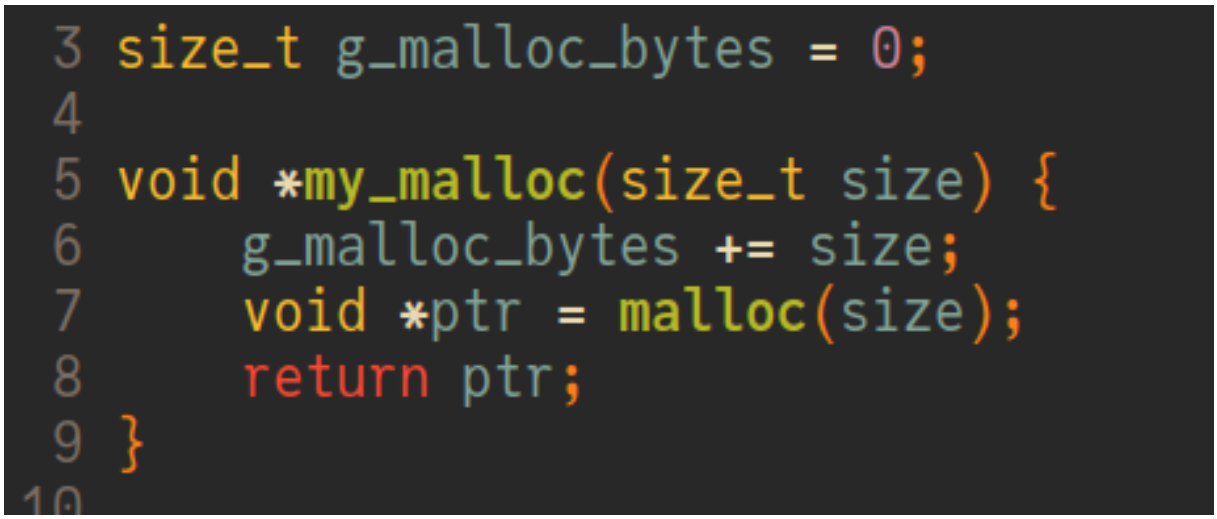
За хранение информации о фильтрах отвечают структуры `Filter`, `FilterExpr`, `FilterTarget`. Есть несколько вариаций фильтрующих выражений, они заданы в перечислении `FilterExprType`. Таким образом поддерживается фильтрация через сравнение, по конкретному значению или по названию поля в узле.

Рисунок 7: Структуры для хранения информации о фильтрах

Сама древовидная структура представлена структурой `Node`, которая реализует связный список из элементов и применяемых к ним фильтров.

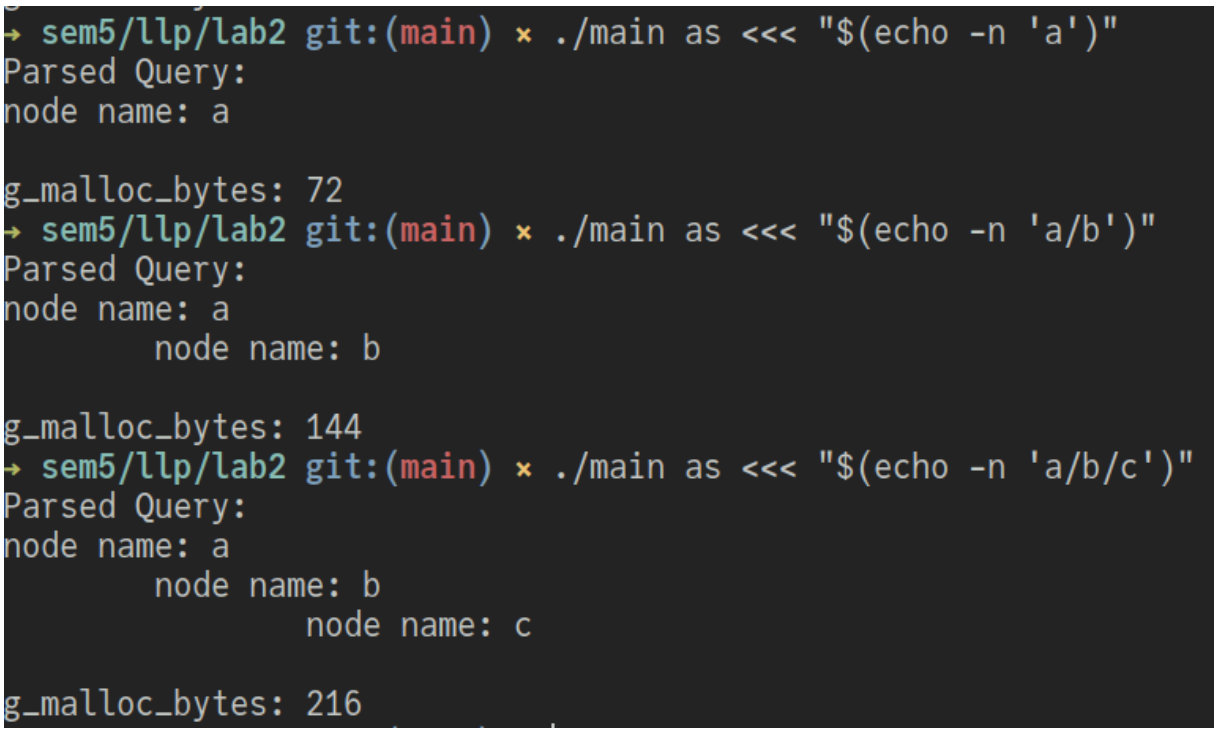
Анализ использования памяти

все аллокации на куче осуществляются через ф-ию `my_malloc`, которая считает суммарное кол-во аллоцируемой памяти.



```
3 size_t g_malloc_bytes = 0;
4
5 void *my_malloc(size_t size) {
6     g_malloc_bytes += size;
7     void *ptr = malloc(size);
8     return ptr;
9 }
10
```

Рисунок 8: Функция my_malloc

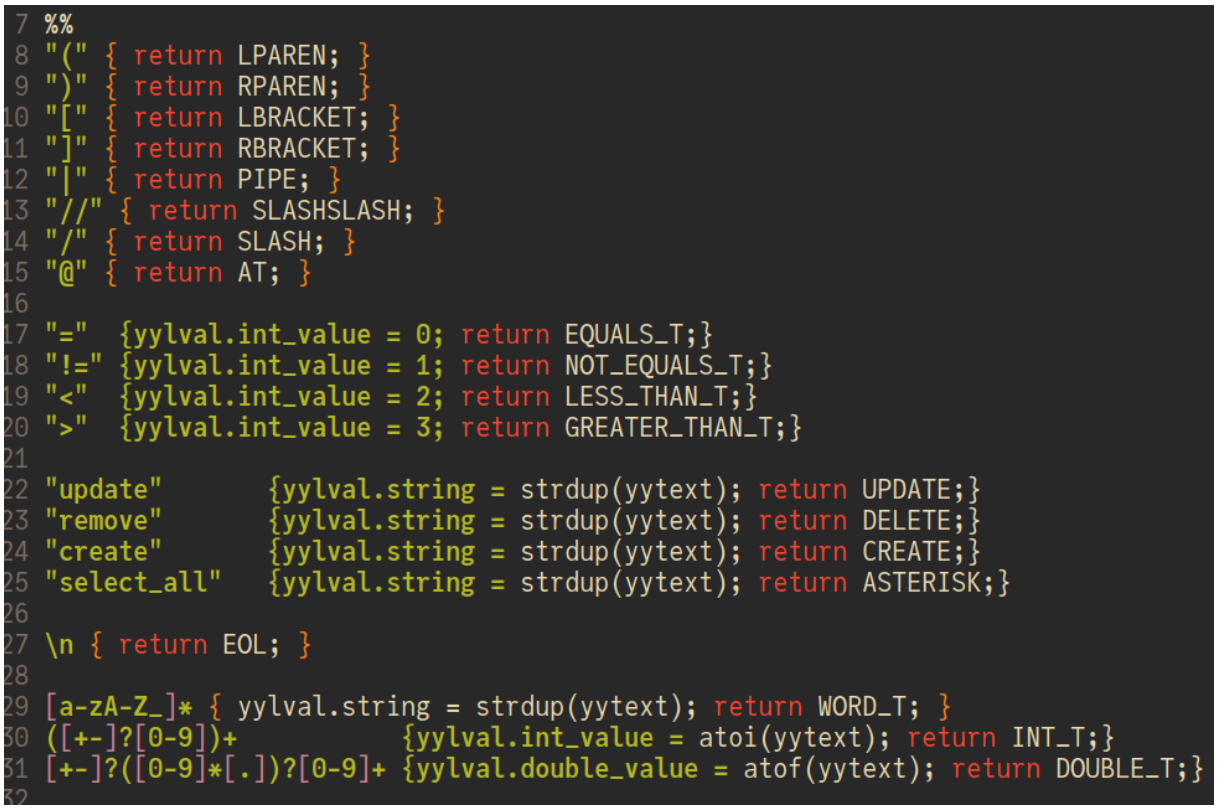


```
→ sem5/llp/lab2 git:(main) × ./main as <<< "$(echo -n 'a')"
Parsed Query:
node name: a

g_malloc_bytes: 72
→ sem5/llp/lab2 git:(main) × ./main as <<< "$(echo -n 'a/b')"
Parsed Query:
node name: a
        node name: b

g_malloc_bytes: 144
→ sem5/llp/lab2 git:(main) × ./main as <<< "$(echo -n 'a/b/c')"
Parsed Query:
node name: a
        node name: b
                node name: c

g_malloc_bytes: 216
```

Рисунок 9: Анализ использования памяти

Программа использует оперативную память только для хранения структуры, и под каждое название переменной выделяется фиксированное кол-во байт.

Токены и Грамматика



```
%%
"(" { return LPAREN; }
")" { return RPAREN; }
"[" { return LBRACKET; }
"]" { return RBRACKET; }
"|" { return PIPE; }
"//" { return SLASHSLASH; }
"/" { return SLASH; }
"@" { return AT; }

"="  {yylval.int_value = 0; return EQUALS_T;}
"!=" {yylval.int_value = 1; return NOT_EQUALS_T;}
"<"  {yylval.int_value = 2; return LESS_THAN_T;}
">"  {yylval.int_value = 3; return GREATER_THAN_T;}

"update"       {yylval.string = strdup(yytext); return UPDATE;}
"remove"       {yylval.string = strdup(yytext); return DELETE;}
"create"       {yylval.string = strdup(yytext); return CREATE;}
"select_all"   {yylval.string = strdup(yytext); return ASTERISK;}

\n { return EOL; }

[a-zA-Z_]* { yylval.string = strdup(yytext); return WORD_T; }
([+-]?[0-9])+          {yylval.int_value = atoi(yytext); return INT_T;}
[+-]?([0-9]*[.])?[0-9]+ {yylval.double_value = atof(yytext); return DOUBLE_T;}
```

Рисунок 10: Пример токенов в лексическом анализаторе

```
function_call
    : function_name LPAREN query RPAREN {
        q.func = get_function_type($1);
    }
    ;

function_name : CREATE | UPDATE | DELETE

filters
    : filter {
        add_filter(&q, $1);
    }
    | filters filter {
        add_filter(&q, $2);
    }
    ;

filter
    : LBRACKET filter_expr RBRACKET {
        $$ = $2;
    }
    | attribute {
        $$ = create_filter_by_var_name($1);
    }
    ;

filter_expr
    : node_value {
        $$ = create_filter_single_value($1);
    }
    | attribute compare_op node_value {
        $$ = create_filter($1, $2, $3);
    }
    ;
```

Рисунок 11: Пример грамматики на Bison

Выводы

В ходе выполнения лабораторной работы изучены Bison и Flex, а также был реализован модуль для разбора запросов XPath.