МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

‘Низкоуровневое программирование’

Вариант 1

Выполнил:

Студент группы P33102

Голиков Д.И.

Преподаватель:

Кореньков Юрий
Дмитриевич

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/Denoske/ITMO-labs/tree/main/3nd%20year/Low%20level%20programming/lab2

Описание работы:

Программа представляет собой консольное приложение, позволяющее принимать на стандартный вход один запрос и выводить результат разбора.

Программа состоит из следующих модулей:

- Printer – Модуль, отвечающий за вывод разобранного запроса.
- Adder - Модуль, отвечающий за создание необходимых структур элементов, также в данном модуле реализованы функции для теста оперативной памяти.
- Decl – Модуль, отвечающий за создание дополнительных структур, которые являются основой разобранного запроса.
- Lex.l – Модуль, в котором описаны лексемы. (flex)
- Parser.y – Модуль, в котором задается грамматика. (bison)

Тестовые примеры работы программы:

```
PS C:\Users\Solnyshko\Desktop\LLP-2\src> mingw32-make --file=Makefile_win all
bison -d parser.y
flex lex.l
gcc main.c decl.c printer.c adder.c lex.yy.c y.tab.c -o main
PS C:\Users\Solnyshko\Desktop\LLP-2\src> .\main
print X-Path query:
/foo[@a=1]
descendant of:
         ---> name: foo
                filter = select property:
                         --> property name a
                                        operator:
                                                equal to
                                                        value = 1

Full ram: 48
```

Рисунок 1 Запуск на Windows

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
/foo[@a=1]
descendant of:
          ---> name: foo
                 filter = select property:
                          --> property name a
                                         operator:
                                                 equal to
                                                         value = 1

Full ram: 56
```

Рисунок 2 Запуск на Linux

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:

/foo[1][@a=1]
descendant of:
         ---> name: foo
                filter = select:
                         --> element number: 1
                                filter = select property:
                                         --> property name a
                                                        operator:
                                                                equal to
                                                                        value = 1
```

Рисунок 3 Вывод первого элемента схемы foo, у которого значение атрибута a = 1

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
/foo[@a=1][@b=2]
descendant of:
        ---> name: foo
               filter = select property:
                        --> property name a
                                        operator:
                                                equal to
                                                        value = 1

                                                filter = select property:
                                                        --> property name b
                                                                        operator:
                                                                                equal to
                                                                                        value = 2
```

Рисунок 4 Вывод всех элементов схемы foo с заданными атрибутами

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
/foo[child][@a=1]
descendant of:
         ---> name: foo
                filter = select:
                         --> descendant schema name: child
                                filter = select property:
                                         --> property name a
                                                         operator:
                                                                 equal to
                                                                         value = 1
```

Рисунок 5 Вывод элементов схемы foo, которые являются предками child и имеют заданный атрибут

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
/foo/@a
descendant of:
          ---> name: foo

                       --> node property: a
```

Рисунок 6 Вывод элемента с именем a в схеме foo

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
foo
root: --> schema name: foo
```

Рисунок 7 Вывод схемы

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
update(/foo[1][@a=1])
descendant of:
         ---> name: foo
                filter = select:
                         --> element number: 1
                                filter = select property:
                                         --> property name a
                                                         operator:
                                                                 equal to
                                                                         value = 1

OPERATION: update
```

Рисунок 8 меняем 1-му элементу схемы foo значение атрибута a=1

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
remove_scheme(foo)
root: --> schema name: foo
OPERATION: remove_scheme
```

Рисунок 9 удалить схему foo (то есть коллекцию foo)

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
remove_scheme(foo/zoo)
root: --> schema name: foo
        descendant of:
                ---> name: zoo
OPERATION: remove_scheme
```

Рисунок 10 Удалить коллекцию zoo, которая является предком foo

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
remove_element(/foo[@a=1])
descendant of:
         ---> name: foo
                filter = select property:
                         --> property name a
                                        operator:
                                                equal to
                                                        value = 1

OPERATION: remove_element
```

Рисунок 11 удалить из элементов схемы foo те, у которых значени атрибута a=1

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
create_scheme(foo[@attrname;type=int32][@attr;type=string])
root: --> schema name: foo
        property name: attrname
        property type: int32
        property name: attr
        property type: string
OPERATION: create_scheme
```

Рисунок 12 создать коллекцию foo с заданными атрибутами и их типами

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
create_element(foo[@a=1])
root: --> schema name: foo
        filter = select property:
                 --> property name a
                                operator:
                                        equal to
                                                value = 1

OPERATION: create_element
```

Рисунок 13 добавить элемент foo с атрибутом

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
create_element(zoo/foo[@a=1])
root: --> schema name: zoo
        descendant of:
                 ---> name: foo
                        filter = select property:
                                 --> property name a
                                                operator:
                                                        equal to
                                                                value = 1

OPERATION: create_element
```

Рисунок 14 добавить дочерний элемент к zoo с именем foo заданным атрибутом

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
select_all
---> print all nodes
```

Рисунок 15 Выбрать всех

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
/foo[a{}]
descendant of:
         ---> name: foo
                filter = select:
                         --> descendant schema name: a
unexpected character
unexpected character
```

Рисунок 16 Ошибка ввода

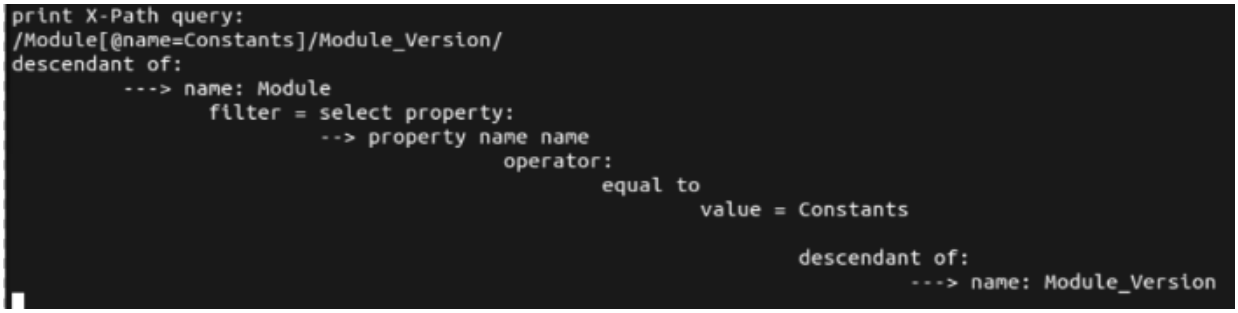

Рисунок 17 Вывести элементы Module_version , которые являются предками Module с условием

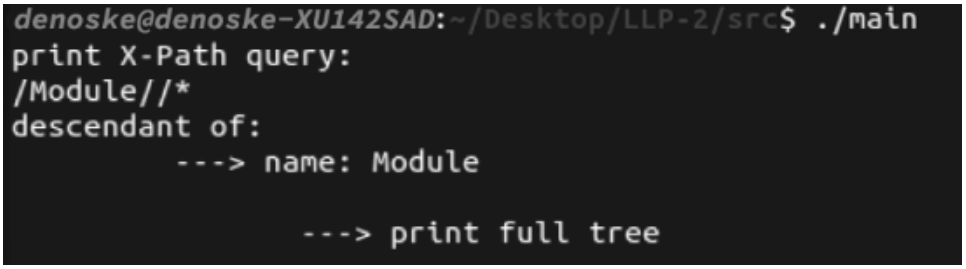

Рисунок 18 Вывести полное дерево предков Module

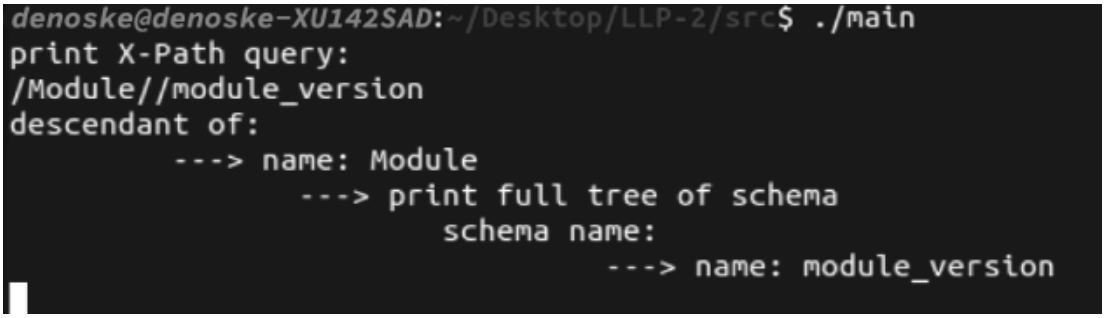

Рисунок 19 Вывести дерево коллекции module_version, которая является предком Module

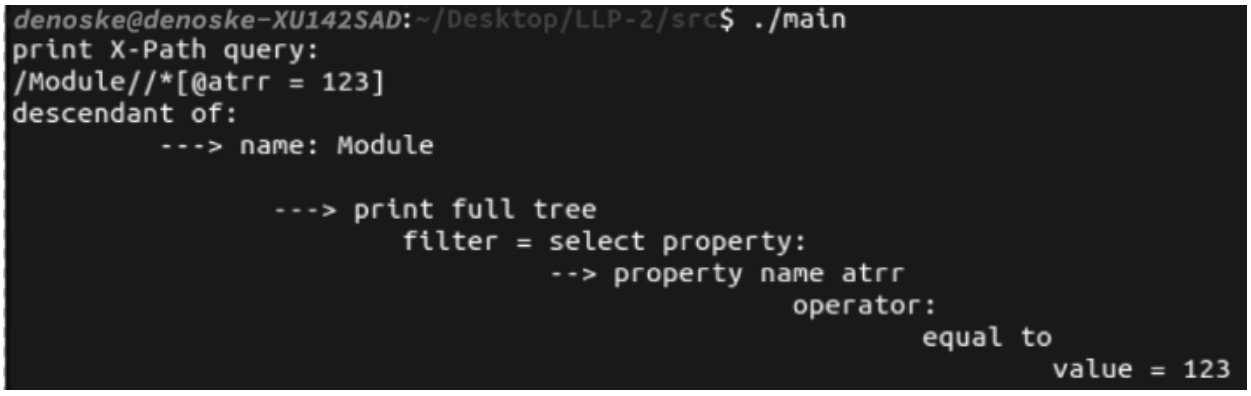

Рисунок 20 Вывести полное дерево по условию

Аспекты реализации:

Внутреннее описание созданной программы:

Для выполнения данного задания синтаксис XPath был немного изменен: Добавлено указание операций. Использование данных операций можно увидеть в предыдущем разделе.

Операции над элементами:

- Update() – Обновление элемента.
- remove_scheme() – Удаление коллекции.
- remove_element() – Удаление элемента.
- create_scheme() – Создание коллекции.
- create_element() – Создание элемента.
- * - Выборка всех элементов.

Операция выборки ноды со всеми узлами не реализована (хотя в xPath присутствует), т.к. на сервере запрос выборки ноды уже подразумевает вывод всех ее атрибутов (не только отношений на уровне док. дерева)

Описание созданных структур:

Первая используемая структура называется element. Она отвечает за создание элемента дерева, который может быть четырех типов – int, bool, string, double. Также в структуре есть вспомогательное поле element_type, которое используется при выводе.

```
struct element {
    union {
        char string_val[30];
        bool boolean;
        int32_t num;
        double double_num;
    };
    int element_type;
};
```

Следующие две структуры являются вспомогательными. Структура filter_scheme хранит в себе объект фильтра, в который входит значение оператора фильтрации, вспомогательный флаг для вывода, значение элемента и значение атрибута, по которому идет фильтрация. Структура property_scheme нужна для хранения объекта свойства, в ней хранится тип и название свойства (используется, например, при создании коллекции).

```
struct filter_scheme {
    int operator_val;
    bool is_single_val;
    struct element* val;
    char attribute[30]; /*optional*/
};
```

```
struct property_scheme {
    int node_type;
    char property_name[30];
};
```

Основной структурой является Filter_object. Она хранит в себе либо фильтр, либо свойство, а также указатель на следующую структуру.

```
struct Filter_object {
    union {
        struct filter_scheme* filter;
        struct property_scheme* prop;
    };
    struct Filter_object* next;
};
```

Анализ использования памяти:
Для анализа использования оперативной памяти были написаны следующие функции:

```
size_t size = 0;

void *test_malloc(size_t size_of){
    size += size_of;
    return malloc(size_of);
}

void *print_ram(){
    printf("%zu\n", size);
}
```

Посмотрим на использование оперативной памяти:

```
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ make all
yacc -d -Wcounterexamples parser.y
flex lex.l
gcc main.c decl.c printer.c adder.c lex.yy.c y.tab.c -o main
denoske@denoske-XU142SAD:~/Desktop/LLP-2/src$ ./main
print X-Path query:
/foo[@a=1]
descendant:
        ---> name: foo
filter = select property:
        --> property name a
operator:
        equal to
                value = 1

Full ram: 56
```

Как можно заметить, программа использует оперативную память только для хранение целевой структуры.

Список самих токенов, написанных благодаря Flex:

```
%{

#include "decl.h"
#include "y.tab.h"
void yyerror (char *s);
%}
%%
"update"                {yylval.str = strdup(yytext); return UPDATE;}
"remove_element"        {yylval.str = strdup(yytext); return REMOVE_EL;}
"remove_scheme"         {yylval.str = strdup(yytext); return REMOVE_SCH;}
"create_element"        {yylval.str = strdup(yytext); return CREATE_EL;}
"create_scheme"         {yylval.str = strdup(yytext); return CREATE_SCH;}

"type=int32"      {yylval.number = 0; return INT32_TYPE;}
"type=double"     {yylval.number = 1; return DOUBLE_TYPE;}
"type=string"     {yylval.number = 2; return STRING_TYPE;}
"type=bool"       {yylval.number = 3; return BOOLEAN_TYPE;}

"true"                          {yylval.bool_value = 1; return BOOL;}
"false"                         {yylval.bool_value = 0; return BOOL;}
([+-]?[0-9])+                   {yylval.number = atoi(yytext); return NUM-
BER;}
```

```
[+-]?([0-9]*[.])?[0-9]+          {yylval.number = atof(yytext); return DOU-
BLE_NUM;}
[a-zA-Z][a-zA-Z0-9_]*             {yylval.str = strdup(yytext); return WORD;}

"="                        {yylval.number = 0; return EQUAL;}
"!="                       {yylval.number = 1; return NOT_EQUAL;}
">"                        {yylval.number = 2; return MORE;}
"<"                        {yylval.number = 3; return LESS;}
"/"                        {return SLASH;}
@                          {return IS_ATTRIBUTE;}
"*"                        {yylval.str = "select all"; return ASTERISK;}
"["                        {return START_FILTER;}
"]"                        {return END_FILTER;}
"("                        {return OPEN_BRACKET;}
")"                        {return CLOSE_BRACKET;}
";"                        {return SEMICOLON;}
\n                        /* skip */;
[ \t]+                    /* skip */;
.               {ECHO; yyerror ("unexpected character");}
%%
```

Пример грамматики, написанной на Bison:

```
node_value:
    NUMBER
    {
        $$ = add_int32_element($1);
    }
    |
     DOUBLE_NUM
     {
        $$ = add_double_element($1);
     }
     |
     WORD
     {
        $$ = add_str_element($1);
     }
     |
     BOOL
     {
        $$ = add_bool_element($1);
     }
     ;

function:
    UPDATE | REMOVE_EL | REMOVE_SCH | CREATE_EL | CREATE_SCH ;

operator:
    EQUAL | NOT_EQUAL | LESS | MORE ;
```

Результаты:

Для выполнения задач было:

- Изучено выбранное средство синтаксического анализа
- Изучен синтаксис языка запросов и записать спецификацию для средства синтаксического анализа.
- Реализован модуль, использующий средство синтаксического анализа для разбора языка запросов.
- Реализована тестовая программа для демонстрации работоспособности созданного модуля, принимающая на стандартный ввод текст запроса и выводящая на стандартный вывод результирующее дерево разбора или сообщение об ошибке.

Выводы:

С помощью YACC&FLEX был реализован модуль производящий синтаксический анализ и разбор запроса XPath.