Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ОТЧЕТ
к лабораторной работе №2
на тему

ЛЕКСИЧЕСКИЙ АНАЛИЗ

Выполнила: студентка гр. 253503
Тимошевич К. С.

Проверил: ассистент кафедры
информатики Гриценко Н. Ю.

Минск 2025

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Цель данной лабораторной работы заключается в разработке лексического анализатора на *Erlang* для подмножества языка программирования *C#*, с использованием которого будет производиться разбор исходных текстов программ [1]. Лексический анализатор должен читать поток символов, представляющих программу, и выделять из него значащие последовательности, называемые лексемами. Эти лексемы могут включать идентификаторы, операторы, константы, знаки математических операций, скобки и другие элементы, характерные для базового синтаксиса языка. Основной задачей является правильное определение лексем, их категоризация и преобразование в понятную структуру, которая может быть использована для дальнейшего синтаксического анализа.

Лексический анализатор должен работать с входными данными, представленными в виде текстового файла, содержащего программу, написанную на заданном подмножестве языка.

В рамках работы нужно создать таблицы констант, ключевых слов, разделителей, логических/математических операторов. Эти таблицы должны быть использованы для поиска ранее встреченных лексем, что позволит избежать дублирования данных. Для констант таблица должна включать информацию о типе (например, переменная с плавающей точкой). При каждом новом обращении к идентификатору нужно проверять его наличие в таблице.

Также необходимо предусмотреть механизм обработки лексических ошибок, которые могут возникать при неверных последовательностях символов, таких как неправильно сформированные идентификаторы, неверные операторы или некорректные константы. Лексический анализатор должен быть способен выявить такие ошибки и сообщить о них, предоставляя пользователю подробную информацию о месте и типе ошибки.

2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

В ходе выполнения лабораторной работы был разработан лексический анализатор для подмножества языка программирования *C#*, предназначенного для работы с выражениями, включающими арифметические операции, скобки, константы и идентификаторы [2]. Задача заключалась в том, чтобы разобрать исходный код, выделить лексемы и классифицировать их по типам: ключевые слова, операторы, разделители, константы, а также выполнить проверки на некоторые лексические ошибки.

Первоначально был создан набор ключевых слов, операторов и разделителей, который позволил корректно идентифицировать основные элементы языка. Ключевые слова, такие как «*if*», «*while*», «*for*», и операторы вроде «+», «-», «*», а также разделители, такие как «;», «,», «(», «)», были занесены в соответствующие списки. Эти элементы впоследствии использовались для того, чтобы на стадии лексического анализа определять, к какому типу принадлежит каждое встреченное слово.

Для проверки значений, встречающихся в программе, был реализован механизм, который проверяет, был ли такой элемент уже занесен в таблицу. Каждое новая константа или оператор, встреченный в коде, проверялись на наличие в таблице. Если элемент не был найден, то добавлялся в таблицу с уникальным индексом.

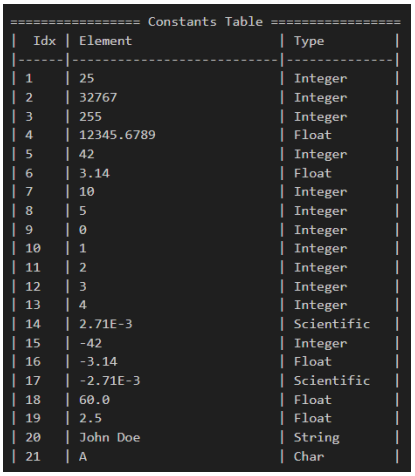
После реализации лексического анализа и обработки таблиц была добавлена функция обработки ошибок. В частности, было предусмотрено распознавание некорректных шаблонов, например, неверных последовательностей символов или нарушений использования строк. Программы анализировали строки на наличие таких ошибок, как отсутствие закрывающих кавычек в строках, некорректные операторы инкремента/декремента и ошибки в идентификаторах.

Также был создан отдельный модуль для обработки строковых и символьных констант в файле, которые извлекались с помощью регулярных выражений. Эти константы добавлялись в таблицу констант с типом «*String*» или «*Char*», в зависимости от типа.

В результате выполнения лабораторной работы был получен лексический анализатор, который выполняет разбор исходных программ, классифицирует лексемы, заполняет таблицы, а также выявляет и сообщает о лексических ошибках в исходном коде [3].

3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

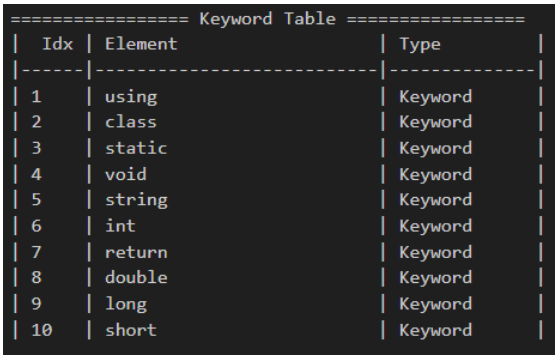
На рисунке 3.1 представлен вывод таблицы констант в результате выполнения программы.



```
================ Constants Table ================
|  Idx | Element                  | Type         |
|------|--------------------------|--------------|
| 1    | 25                       | Integer      |
| 2    | 32767                    | Integer      |
| 3    | 255                      | Integer      |
| 4    | 12345.6789               | Float        |
| 5    | 42                       | Integer      |
| 6    | 3.14                     | Float        |
| 7    | 10                       | Integer      |
| 8    | 5                        | Integer      |
| 9    | 0                        | Integer      |
| 10   | 1                        | Integer      |
| 11   | 2                        | Integer      |
| 12   | 3                        | Integer      |
| 13   | 4                        | Integer      |
| 14   | 2.71E-3                  | Scientific   |
| 15   | -42                      | Integer      |
| 16   | -3.14                    | Float        |
| 17   | -2.71E-3                 | Scientific   |
| 18   | 60.0                     | Float        |
| 19   | 2.5                      | Float        |
| 20   | John Doe                 | String       |
| 21   | A                        | Char         |
```

Рисунок 3.1 – Таблица констант

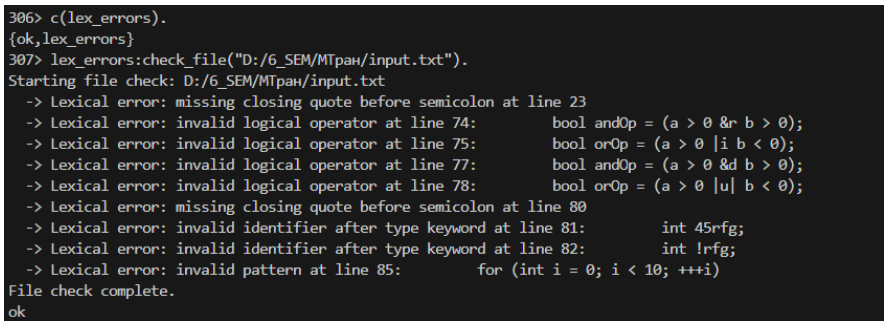
На рисунке 3.2 показана часть вывода таблицы с ключевыми словами.



```
================= Keyword Table =================
|  Idx | Element                  | Type         |
|------|--------------------------|--------------|
| 1    | using                    | Keyword      |
| 2    | class                    | Keyword      |
| 3    | static                   | Keyword      |
| 4    | void                     | Keyword      |
| 5    | string                   | Keyword      |
| 6    | int                      | Keyword      |
| 7    | return                   | Keyword      |
| 8    | double                   | Keyword      |
| 9    | long                     | Keyword      |
| 10   | short                    | Keyword      |
```

Рисунок 3.2 – Таблица ключевых слов

Пример обработки ошибок и вывода сообщений о них представлен на рисунке 3.3.



```
306> c(lex_errors).
{ok,lex_errors}
307> lex_errors:check_file("D:/6_SEM/МТран/input.txt").
Starting file check: D:/6_SEM/МТран/input.txt
  -> Lexical error: missing closing quote before semicolon at line 23
  -> Lexical error: invalid logical operator at line 74:         bool andOp = (a > 0 &r b > 0);
  -> Lexical error: invalid logical operator at line 75:         bool orOp = (a > 0 |i b < 0);
  -> Lexical error: invalid logical operator at line 77:         bool andOp = (a > 0 &d b > 0);
  -> Lexical error: invalid logical operator at line 78:         bool orOp = (a > 0 |u| b < 0);
  -> Lexical error: missing closing quote before semicolon at line 80
  -> Lexical error: invalid identifier after type keyword at line 81:         int 45rfg;
  -> Lexical error: invalid identifier after type keyword at line 82:         int !rfg;
  -> Lexical error: invalid pattern at line 85:         for (int i = 0; i < 10; +++i)
File check complete.
ok
```

Рисунок 3.3 – Сообщения об ошибках

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы был разработан лексический анализатор на языке *Erlang* для подмножества языка программирования *C#* [4]. Разработанный анализатор выполняет разбор входного кода, выделяет ключевые лексемы, включая ключевые слова, операторы, разделители и константы, а также формирует соответствующие таблицы для их хранения и обработки.

Для организации хранения данных были созданы таблицы констант, ключевых слов, операторов и разделителей, что позволило минимизировать дублирование информации и упростить обработку лексем. Также реализован механизм проверки наличия элементов в таблице перед их добавлением, что способствует оптимизации работы анализатора.

Дополнительно были реализованы модули для обработки строковых и символьных констант с использованием регулярных выражений, что позволило корректно обрабатывать текстовые значения и добавлять их в таблицу констант

Реализованный алгоритм анализа позволил не только классифицировать элементы кода, но и выявлять лексические ошибки. В частности, были обработаны такие ошибки, как отсутствие закрывающих кавычек в строковых литералах, некорректные логические операторы, ошибки в именах идентификаторов и неверные шаблоны операторов инкремента. Анализатор предоставляет детализированные сообщения об ошибках, указывая их местоположение в коде, что упрощает процесс отладки.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Erlang/OTP – documentation [Электронный ресурс]. – Режим доступа: https://www.erlang.org/. – Дата доступа: 01.02.2025.

[2] Основы C# [Электронный ресурс]. – Режим доступа: https://code-basics.com/ru/languages/csharp. – Дата доступа: 01.02.2025.

[3] Лексический анализ в разработке компилятора с примером [Электронный ресурс]. – Режим доступа: https://www.guru99.com/ru/compiler-design-lexical-analysis.html. – Дата доступа: 02.02.2025.

[4] Редкие языки: Erlang. Что за зверь и зачем нужен [Электронный ресурс]. – https://gb.ru/posts/erlang. – Дата доступа: 02.02.2025.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
-module(mtran).
-export([read_file/0, hello/0, init_tables/0, print_tables/0,
process_file/1]).

operators() ->
    ["+", "-", "*", "/", "%", "&&", "||", "!", "==", "!=", ">", "<", ">=",
"<="].

delimiters() ->
    [";", ",", ".", "(", ")", "{", "}", "[", "]"].

is_constant(Token) ->
    case re:run(Token, "^[+-]?[0-9]+$", [{capture, none}]) of
        match -> {true, "Integer"};
        nomatch -> case re:run(Token, "^[+-]?[0-9]+(\\.[0-9]+)?$", [{capture,
none}]) of
            match -> {true, "Float"};
            nomatch -> case re:run(Token, "^[+-]?[0-9]+(\\.[0-9]+)?([eE][-
+]?[0-9]+)$", [{capture, none}]) of
                match -> {true, "Scientific"};
                nomatch -> {false, ""}
            end
        end
    end.

init_tables() ->
    lists:foreach(fun(Table) ->
        case ets:info(Table) of
            undefined -> ets:new(Table, [named_table, set, public]);
            _ -> ets:delete_all_objects(Table)
        end
    end, [names_table, operators_table, delimiters_table, constants_table]),
    io:format("Tables initialized and cleared.~n").


process_file(File) ->
    case file:read_file(File) of
        {ok, Content} ->
            String = erlang:binary_to_list(Content),
            Tokens = re:split(String, "([ \t\n\r;,(){}\\[\\]])", [{return,
list}, trim]),
            process_tokens(Tokens),
            StringConstants = test1:parse_strings_from_file(File),
            lists:foreach(fun(Str) -> insert_token(constants_table, Str,
"String") end, StringConstants),
            CharConstants = test1:parse_chars_from_file(File),
            lists:foreach(fun(Char) -> insert_token(constants_table, Char,
"Char") end, CharConstants),
            print_tables();
        {error, Reason} ->
            io:format("Error reading file ~s: ~s~n", [File,
atom_to_list(Reason)])
    end.

process_tokens([]) -> ok;
process_tokens([Token | Rest]) ->
    case lists:member(Token, cs_keywords()) of
        true -> insert_token(names_table, Token, "Keyword");
```

```
        false ->
            case lists:member(Token, operators()) of
                true -> insert_token(operators_table, Token, "Operator");
                false ->
                    case lists:member(Token, delimiters()) of
                        true -> insert_token(delimiters_table, Token,
"Delimiter");
                        false ->
                            case is_constant(Token) of
                                {true, Type} -> insert_token(constants_table,
Token, Type);
                                {false, _} -> ok
                            end
                    end
            end
    end,
    process_tokens(Rest).

insert_token(Table, Token, Type) ->
    case ets:lookup(Table, Token) of
        [] ->
            Index = length(ets:tab2list(Table)) + 1,
            ets:insert(Table, {Token, Index, Type}),
            io:format("Added to ~s table: {~s, ~p, ~s}~n",
[atom_to_list(Table), Token, Index, Type]);
        _ -> ok
    end.

print_tables() ->
    print_table(names_table, "Keyword Table"),
    print_table(operators_table, "Operators Table"),
    print_table(delimiters_table, "Delimiters Table"),
    print_table(constants_table, "Constants Table").

print_table(Table, Title) ->
    io:format("================= ~s =================~n", [Title]),
    TableList = ets:tab2list(Table),
    Sorted = lists:sort(fun({_, Index1, _}, {_, Index2, _}) -> Index1 =<
Index2 end, TableList),
    io:format("| ~4s | ~-25s | ~-12s |~n", ["Idx", "Element", "Type"]),
    io:format("|~s|~s|~s|~n",
              [lists:duplicate(6, "-"),
               lists:duplicate(27, "-"),
               lists:duplicate(14, "-")]),
    lists:foreach(fun({Token, Index, Type}) ->
                  io:format("| ~4p | ~-25s | ~-12s |~n", [Index, Token, Type])
                   end, Sorted),
    io:format("~n", []).


read_file() ->
    File = "D:/6_SEM/МТран/input.txt",
    case file:read_file(File) of
        {ok, Content} ->
            io:format("Content type: ~p~n", [Content]),
            String = erlang:binary_to_list(Content),
            io:format("File content: ~s~n", [String]);
        {error, Reason} ->
            io:format("Error reading file ~s: ~s~n", [File,
atom_to_list(Reason)])
    end.

hello() ->
    io:format("Hello, Erlang!~n").
```