

Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ  
И РАДИОЭЛЕКТРОНИКИ  
КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе № 2

По теме «Лексический анализ»

Выполнил:  
студент гр. 053502  
Герчик А. В.

Проверил:  
Гриценко Н. Ю.

Минск 2023

## Содержание

1. Цель работы	3
2. Результаты	4
2.1 Результаты выделения токенов	4
2.2 Нахождение и локализация ошибок	6
Приложение. Код программ	7
1. Тестовая программа	7
2. Лексический анализатор	8

## **1. Цель работы**

Разработка лексического анализатора подмножества язык программирования, определенного в лабораторной работе 1.

## 2. Результаты

В ходе выполнения работы, удалось сделать лексический анализатор подмножества языка программирования, определенного в лабораторной работе 1.

### 2.1 Результаты выделения токенов

Для тестовой программы(см. приложение), из ЛР1 вывод анализатора имеет следующий вид:

На рисунке 1 показан вывод используемых типов переменных в тестовой программе.

```
int - variable type  
float - variable type
```

Рисунок 1: Вывод используемых типов

На рисунке 2 показан вывод используемых ключевых слов.

```
#include - key word  
<iostream> - key word  
<math.h> - key word  
using - key word  
namespace - key word  
std - key word  
cout - key word  
cin - key word  
switch - key word  
case - key word  
break - key word  
default - key word  
endl - key word  
return - key word
```

Рисунок 2: Вывод используемых ключевых слов

На рисунке 3 показан вывод используемых операторов.

```
= - operator  
* - operator  
- - operator  
+ - operator
```

Рисунок 3: Вывод используемых операторов

На рисунке 4 показан вывод используемых переменных

```
a - VAR  
b - VAR  
c - VAR  
Y - VAR  
n - VAR
```

Рисунок 4. Вывод используемых переменных

## 2.2 Нахождение и локализация ошибок

Для тестовой программы(см. приложение), из ЛР1 вывод анализатора, а именно поиска ошибок, имеет следующий вид:

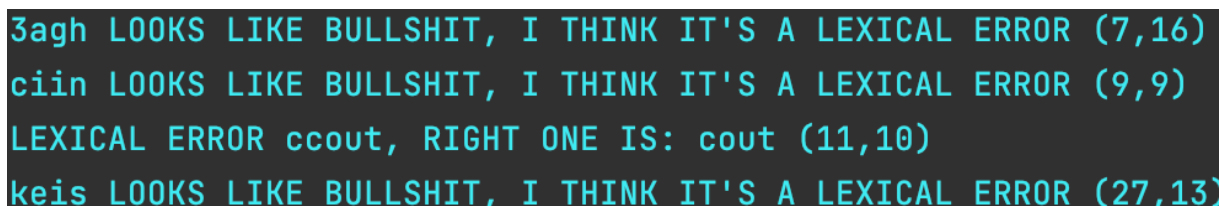
1) `int n, 3agh;` В данной строке имеется лексическая ошибка, а именно имя переменной не может начинаться с цифры.

2) `ciin >> n;` В данной строке имеется лексическая ошибка, а именно ключевое слово `ciin` является ошибочным.

3) `ssout << n;` В данной строке имеется лексическая ошибка, а именно ключевое слово `ssout` является ошибочным. В данном случае, анализатор выдает предполагаемое верное значение.

4) `keis 56 : break;` В данной строке имеется лексическая ошибка, а именно ключевое слово `keis` является ошибочным.

На рисунке 5 показан вывод найденных лексических ошибок.



```
3agh LOOKS LIKE BULLSHIT, I THINK IT'S A LEXICAL ERROR (7,16)
ciin LOOKS LIKE BULLSHIT, I THINK IT'S A LEXICAL ERROR (9,9)
LEXICAL ERROR ccout, RIGHT ONE IS: cout (11,10)
keis LOOKS LIKE BULLSHIT, I THINK IT'S A LEXICAL ERROR (27,13)
```

Рисунок 5. Вывод найденных лексических ошибок

# Приложение. Код программ

## 1. Тестовая программа

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    float a, b, c, Y = 0.0;
    int n, 3agh;

    ciin >> n;

    ccout << n;

    cout << "Enter a : ";
    cin >> a;
    cout << "Enter b : ";
    cin >> b;
    cout << "Enter c : ";
    cin >> c;
    cout << "Enter N : ";
    cin >> n;

    switch (n)
    {
    case 2:
        Y = (b * c) - pow(a, 2);
        break;
        keis 56 : break;
    case 56:
        Y = (b * c);
        break;
    case 7:
        Y = pow(a, 2) + c;
        break;
    case 3:
```

```

        Y = a - (b * c);
        break;
default:
    Y = pow((a + b), 3);
    break;
}

cout << "Y is : " << Y << endl;
return 0;
}

```

## 2. Лексический анализатор

```

namespace LAB2;

public static class Program
{
    public static void Main()
    {
        var varTypesDict = new Dictionary<string, string>()
        {
            { "int", "variable type" },
            { "float", "variable type" },
            { "char", "variable type" },
            { "string", "variable type" }
        };
        var keywordsDict = new Dictionary<string, string>()
        {
            { "const", "key word" },
            { "if", "key word" },
            { "else", "key word" },
            { "else if", "key word" },
            { "do", "key word" },
            { "while", "key word" },
            { "for", "key word" },
            { "return", "key word" },
            { "#include", "key word" },
            { "<iostream>", "key word" },
            { "<math.h>", "key word" },

```



```

        { "using", "key word" },
        { "namespace", "key word" },
        { "std", "key word" },
        { "break", "key word" },
        { "continue", "key word" },
        { "case", "key word" },
        { "switch", "key word" },
        { "cin", "key word" },
        { "cout", "key word" },
        { "default", "key word" },
        { "endl", "key word" }
    };

    var opertorsDict = new Dictionary<string, string>()
    {
        { "-", "operator" },
        { "+=", "operator" },
        { "=", "operator" },
        { "+", "operator" },
        { "==", "operator" },
        { "%", "operator" },
        { "*", "operator" }
    };

    var functionsDict = new Dictionary<string, string>()
    {
        { "main", "function name" },
        { "pow", "function name" }
    };

    var fileAsList = File.ReadAllLines("test.cpp");

    for (var i = 0; i < fileAsList.Length; i++) fileAsList[i] += "\n";

    //string fileText = File.ReadAllText("test.cpp");
    var counter = 0;
    //Console.WriteLine(fileText);

    var oneWord = "";

```

```

var stringStart = false;
var variable = false;

var foundVarTypes = new List<string>();
var foundKeywords = new List<string>();
var foundOperators = new List<string>();
var foundVars = new List<string>();
var errors = new List<string>();

for (var index = 0; index < fileAsList.Length; index++)
{
    counter = 0;
    foreach (var character in fileAsList[index])
    {
        counter++;
        if (character == '"' && !stringStart)
        {
            stringStart = true;
        }
        else if (character == '"' && stringStart && oneWord.Length > 1)
        {
            oneWord = "";
            stringStart = false;
            continue;
        }

        if (variable)
        {
            if (character == Convert.ToChar(32)) continue;

            if (character == Convert.ToChar(","))
            {
                if (char.IsDigit(oneWord[0]))
                {
                    Console.WriteLine(
                        $"{oneWord} LOOKS LIKE BULLSHIT, I THINK IT'S
A LEXICAL ERROR ( {index + 1}, {counter} )");
                    errors.Add(

```

```

        $"{oneWord} LOOKS LIKE BULLSHIT, I THINK IT'S
A LEXICAL ERROR ({index + 1},{counter}))";
    }
    else
    {
        Console.WriteLine($"{oneWord} - VAR");
        foundVars.Add($"{oneWord} - VAR");
        oneWord = "";
        continue;
    }
}
else if (character == Convert.ToChar(",") || character ==
Convert.ToChar("="))
{
    if (char.IsDigit(oneWord[0]))
    {
        Console.WriteLine(
            $"{oneWord} LOOKS LIKE BULLSHIT, I THINK IT'S
A LEXICAL ERROR ({index + 1},{counter}))";
        errors.Add(
            $"{oneWord} LOOKS LIKE BULLSHIT, I THINK IT'S
A LEXICAL ERROR ({index + 1},{counter}))";
        variable = false;
        oneWord = "";
        continue;
    }
    else
    {
        Console.WriteLine($"{oneWord} - VAR");
        foundVars.Add($"{oneWord} - VAR");
        oneWord = "";
        variable = false;
        continue;
    }
}
}
}

```

```

oneWord += character;
if (!stringStart)
    if (character == Convert.ToChar("\n") ||
        character == Convert.ToChar(32) ||
        character == Convert.ToChar(33) ||
        //character == Convert.ToChar(34) ||
        // character == Convert.ToChar(35) ||
        character == Convert.ToChar(36) ||
        // character == Convert.ToChar(37) ||
        character == Convert.ToChar(38) ||
        character == Convert.ToChar(39) ||
        character == Convert.ToChar(40) ||
        character == Convert.ToChar(41) ||
        // character == Convert.ToChar(42) ||
        // character == Convert.ToChar(43) ||
        character == Convert.ToChar(44) ||
        // character == Convert.ToChar(45) ||
        // character == Convert.ToChar(46) ||
        character == Convert.ToChar(47) ||
        character == Convert.ToChar(58) ||
        character == Convert.ToChar(59) ||
        // character == Convert.ToChar(60) ||
        // character == Convert.ToChar(61) ||
        // character == Convert.ToChar(62) ||
        character == Convert.ToChar(63) ||
        character == Convert.ToChar(64) ||
        character == Convert.ToChar(91) ||
        character == Convert.ToChar(92) ||
        character == Convert.ToChar(93) ||
        character == Convert.ToChar(94) ||
        character == Convert.ToChar(95) ||
        character == Convert.ToChar(96) ||
        character == Convert.ToChar(123) ||
        character == Convert.ToChar(124) ||
        character == Convert.ToChar(125) ||
        character == Convert.ToChar(126))
    {
        var lastCharacter = oneWord[oneWord.Length - 1];

```

```

        oneWord = oneWord.Remove(oneWord.Length - 1);
        if (oneWord.Length >= 1 && oneWord[0] ==
Convert.ToChar("\n")) oneWord = oneWord.Remove(0);

```

```

        if (keywordsDict.ContainsKey(oneWord))
        {
            Console.WriteLine($"{oneWord} -
{keywordsDict[oneWord]}");
            foundKeywords.Add($"{oneWord} -
{keywordsDict[oneWord]}");
            oneWord = "";
            continue;
        }

```

```

        if (varTypesDict.ContainsKey(oneWord))
        {
            Console.WriteLine($"{oneWord} -
{varTypesDict[oneWord]}");
            foundVarTypes.Add($"{oneWord} -
{varTypesDict[oneWord]}");
            oneWord = "";
            variable = true;
            continue;
        }

```

```

        if (opertorsDict.ContainsKey(oneWord))
        {
            Console.WriteLine($"{oneWord} -
{opertorsDict[oneWord]}");
            foundOperators.Add($"{oneWord} -
{opertorsDict[oneWord]}");
            oneWord = "";
            continue;
        }

```

```

        if (functionsDict.ContainsKey(oneWord))
        {

```

```

        Console.WriteLine($"{oneWord} -
{functionsDict[oneWord]}");
        oneWord = "";
        variable = false;
        continue;
    }

    if (oneWord.Length > 1)
    {
        foreach (var key in keywordsDict.Keys)
            if (oneWord.Contains(key))
            {
                Console.WriteLine(
                    $"LEXICAL ERROR {oneWord}, RIGHT ONE
IS: {key} ({index + 1},{counter})");
                errors.Add($"LEXICAL ERROR {oneWord}, RIGHT
ONE IS: {key} ({index + 1},{counter})");
                oneWord = "";
                break;
            }

        foreach (var key in varTypesDict.Keys)
            if (oneWord.Contains(key))
            {
                Console.WriteLine(
                    $"LEXICAL ERROR {oneWord}, RIGHT ONE
IS: {key} ({index + 1},{counter})");
                errors.Add($"LEXICAL ERROR {oneWord}, RIGHT
ONE IS: {key} ({index + 1},{counter})");
                oneWord = "";
                break;
            }

        if (oneWord != ">>" && oneWord != "<<" && oneWord !
= Convert.ToString(32) &&
            !int.TryParse(oneWord, out var a) && !
float.TryParse(oneWord, out var b) &&
            oneWord != Convert.ToString(""))

```

```

        {
            Console.WriteLine(
                $"{oneWord} LOOKS LIKE BULLSHIT, I THINK
IT'S A LEXICAL ERROR ({index + 1},{counter})");
            errors.Add(
                $"{oneWord} LOOKS LIKE BULLSHIT, I THINK
IT'S A LEXICAL ERROR ({index + 1},{counter})");
        }
    }

    oneWord = "";
}
}
}

```

```

Console.WriteLine();

```

```

foundVarTypes = foundVarTypes.Distinct().ToList();
foundKeywords = foundKeywords.Distinct().ToList();
foundOperators = foundOperators.Distinct().ToList();
foundVars = foundVars.Distinct().ToList();
errors = errors.Distinct().ToList();

```

```

Console.WriteLine("-----");

```

```

Console.WriteLine();
foreach (var element in foundVarTypes) Console.WriteLine(element);

```

```

Console.WriteLine();

```

```

Console.WriteLine();
foreach (var element in foundKeywords) Console.WriteLine(element);

```

```

Console.WriteLine();

```

```

Console.WriteLine();
foreach (var element in foundOperators) Console.WriteLine(element);

```

```
Console.WriteLine();
```

```
Console.WriteLine();
```

```
foreach (var element in foundVars) Console.WriteLine(element);
```

```
Console.WriteLine();
```

```
Console.WriteLine();
```

```
foreach (var element in errors) Console.WriteLine(element);
```

```
Console.WriteLine();
```

```
Console.WriteLine("-----");
```

```
}
```

```
}
```