Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

Лексический анализ.

Выполнил:

Студент гр. 053502

Юрьев В.А.

Проверил:

Ассистент кафедры информатики

Гриценко Н.Ю.

Минск 2023

**Содержание**

[1. Цель работы 3](#_Toc128448402)

[2. Результаты 4](#_Toc128448403)

[2.1. Выделение лексем (токенов) 4](#_Toc128448404)

[2.2. Обнаружение и локализация ошибок 5](#_Toc128448405)

[Приложение 7](#_Toc128448406)

[1. Тестовая программа 7](#_Toc128448407)

[2. Лексический анализатор 7](#_Toc128448408)

# **Цель работы**

Разработка лексического анализатора подмножества языка программирования, определенного в лабораторной работе 1. Определяются лексические правила. Выполняется перевод потока символов в поток лексем (токенов).

# **Результаты**

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

## **Выделение лексем (токенов)**

Для тестовой программы(см. приложение) был получен следующий результат:

На рисунке 1 показаны используемые в тестовой программе ключевые слова.

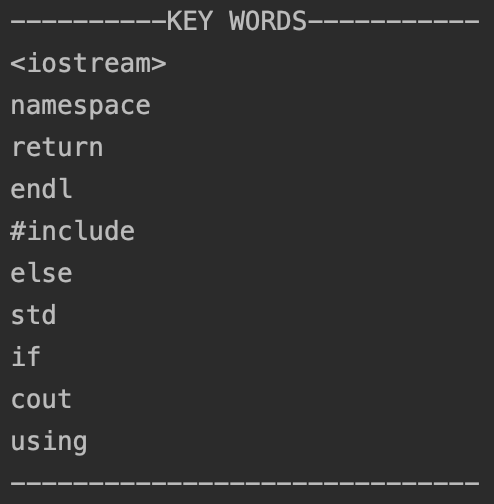


Рисунок 1. Вывод ключевых слов

На рисунке 2 показаны используемые в тестовой программе константы.

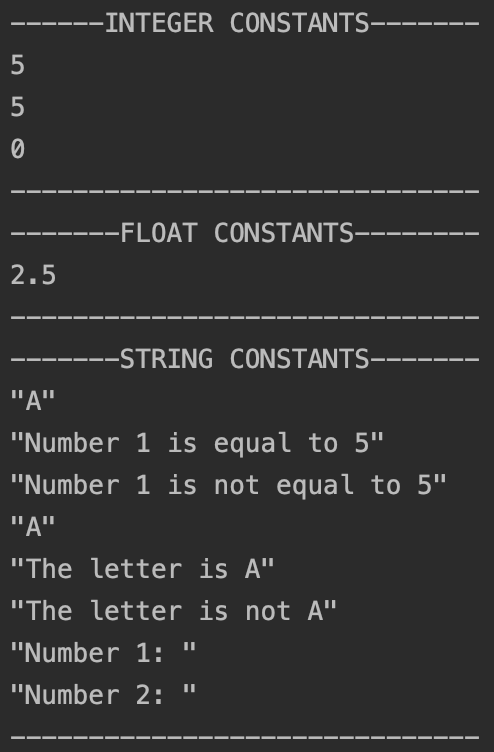


Рисунок 2. Вывод констант

На рисунке 3 показаны используемые в тестовой программе операторы.

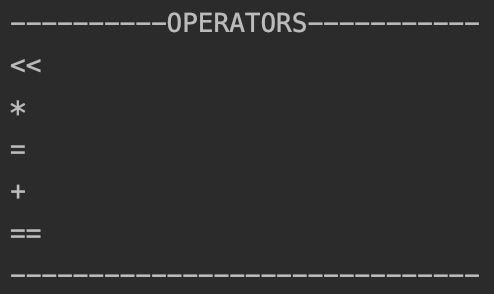


Рисунок 3. Вывод операторов

На рисунке 4 показаны используемые в тестовой программе типы переменных.

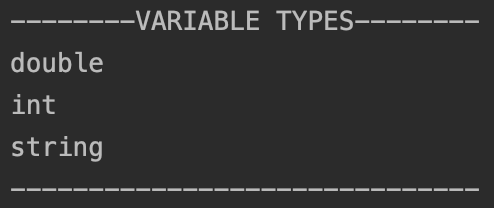


Рисунок 4. Вывод типов переменных

На рисунке 5 показаны используемые в тестовой программе переменные.

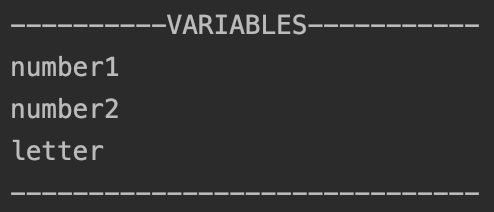


Рисунок 5. Вывод переменных

## **Обнаружение и локализация ошибок**

Для тестовой программы(см. приложение) результат работы анализатор следующий:

1) int 4number. В данной строке лексическая ошибка. Имя переменной может начинаться только с буквы либо символа \_.

2) iff (letter == “A”). В данной строке лексическая ошибка. Ключевое слово iff ошибочно, вместо него анализатором предполагается правильное if.

3) number1 = number1 + \* number1. В данной строке лексическая ошибка, так как после оператора + идет оператор \*.

4) В данной строке ошибка, так как отсутствует закрывающая скобка блока main.

На рисунке 6 показаны обнаруженные в тестовой программе ошибки.

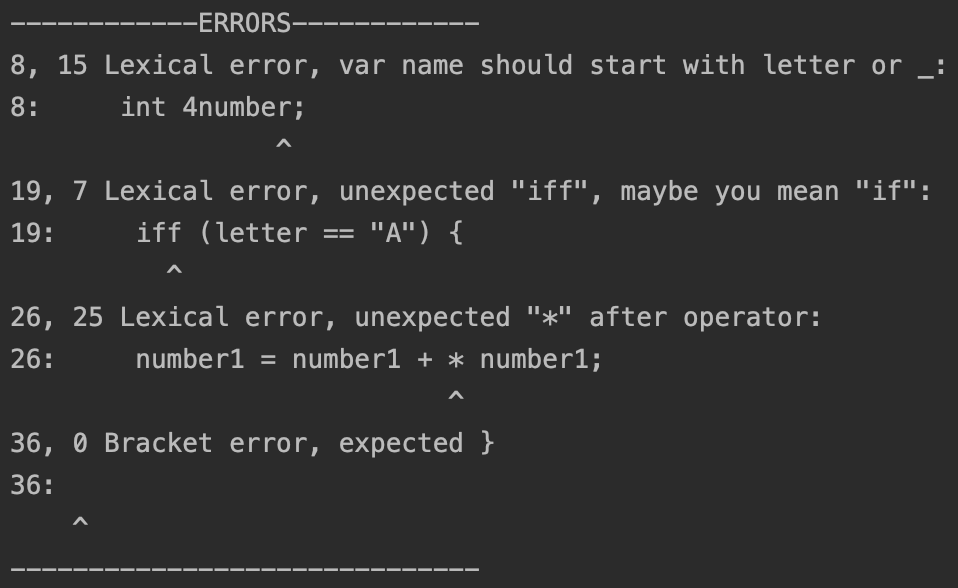


Рисунок 6. Вывод обнаруженных ошибок

# **Приложение**

## **Тестовая программа**

1. #include <iostream>
2. using namespace std;
3. int main() {
4. int number1 = 5;
5. double number2 = 2.5;
6. string letter = "A";
7. int 4number;
9. if number1 == 5 {
10. cout << "Number 1 is equal to 5";
11. }
12. else {
13. cout << "Number 1 is not equal to 5";
14. }
16. cout << endl;
18. iff (letter == "A") {
19. cout << "The letter is A";
20. }
21. else {
22. cout << "The letter is not A";
23. }
25. number1 = number1 + \* number1;
26. cout << endl;
28. cout << "Number 1: " << number1 << endl
29. << "Number 2: " << number2 << endl;
31. return 0;

## **Лексический анализатор**

var\_types = ['bool', 'char', 'int', 'float', 'double', 'string']  
key\_words = ['#include', '<iostream>', '<math.h>', 'using', 'namespace',  
 'std', 'for', 'while', 'do', 'continue', 'break',  
 'if', 'else', 'switch', 'case', 'return',  
 'cin', 'cout', 'endl', 'default']  
operators = ['+', '-', '\*', '/', '%', '=', '==', '+=', '<<', '>>', '<', '>', '&&', '||', '&', '|']  
functions = ['main', 'factorial', 'pow', 'abs']  
brackets = {  
 '(': ')',  
 '{': '}',  
 '[': ']'  
}  
  
  
def bracket\_check(lines):  
 stack = []  
 for row in range(len(lines)):  
 for column in range(len(lines[row])):  
 s = lines[row][column]  
 if s in brackets.keys():  
 stack.append(s)  
 elif s in brackets.values():  
 if len(stack) == 0 or brackets[stack[-1]] != s:  
 return f'{row}, {column} Bracket error, expected {brackets[stack[-1]]}\n' \  
 f'{row + 1}: {lines[row]}\n' \  
 f'{" " \* (len((row + 1).\_\_str\_\_()) + 1 + column)} ^'  
 else:  
 stack.pop()  
  
 if len(stack) != 0:  
 return f'{len(lines)}, 0 Bracket error, expected {brackets[stack[-1]]}\n' \  
 f'{len(lines)}: {lines[len(lines) - 1]}\n' \  
 f'{" " \* (len((len(lines) - 1).\_\_str\_\_()) + 1)} ^'  
  
 return True  
  
  
def is\_int(number\_str):  
 if not number\_str:  
 return False  
 if not (number\_str.isdigit() or (number\_str[0] == '-' and number\_str[1:].isdigit())):  
 return False  
 number = int(number\_str)  
 return number\_str == str(number)  
  
  
def is\_float(number\_str):  
 if not number\_str:  
 return False  
 try:  
 float(number\_str)  
 except ValueError:  
 return False  
 return not (is\_int(number\_str) or (number\_str[0] == '-' and is\_int(number\_str[1:])))  
  
  
def whitespaces():  
 print(30 \* '-')  
  
  
def whitespace\_check(symbol):  
 return symbol == ' ' or symbol == '\t' or symbol == ';'  
  
  
def operator\_error(current, line, row, column):  
 return (f'{row}, {column} Lexical error, unexpected "{current}" after operator:\n'  
 f'{row}: {line}\n'  
 f'{" " \* (len((row + 1).\_\_str\_\_()) + 1 + column)}^')  
  
  
def var\_name\_error(line, row, column):  
 return (f'{row}, {column} Lexical error, var name should start with letter or \_:\n'  
 f'{row}: {line}\n'  
 f'{" " \* (len((row + 1).\_\_str\_\_()) + 1 + column)}^')  
  
  
def lexical\_error(current: str, line, row, column):  
 mean = ''  
 for value in var\_types + key\_words + functions:  
 if current.\_\_contains\_\_(value):  
 mean = f', maybe you mean "{value}"'  
 break  
 return (f'{row}, {column} Lexical error, unexpected "{current}"{mean if mean != "" else ""}:\n'  
 f'{row}: {line}\n'  
 f'{" " \* (len((row + 1).\_\_str\_\_()) + 1 + column)}^')  
  
  
def token\_output(name: str, token\_collection):  
 length = 30 - len(name)  
 print('-' \* math.floor(length / 2) + name.upper() + '-' \* math.ceil(length / 2))  
 for token in token\_collection:  
 print(token)  
 whitespaces()

def get\_file():  
 filetypes = (  
 ('C++ files', '\*.cpp'),  
 )  
 path = filedialog.askopenfilename(filetypes=filetypes)  
 with open(path, 'r') as f:  
 data = f.read()  
  
 return data  
  
  
def main():  
 lines = get\_file().split('\n')  
 var\_tokens = {}  
  
 integer\_tokens = []  
 float\_tokens = []  
 string\_tokens = []  
  
 var\_types\_tokens = []  
 key\_word\_tokens = []  
 operator\_tokens = []  
  
 errors = []  
  
 for row in range(len(lines)):  
 current = ''  
 prev\_var\_type = ''  
 prev\_is\_operator = False  
 string\_started = False  
 for column in range(len(lines[row])):  
 s = lines[row][column]  
 if column == len(lines[row]) - 1 and not (  
 whitespace\_check(s)  
 or s in brackets.keys() or s in brackets.values()  
 or s == ',' or s == '"' or s == ":"):  
 current += s  
  
 if string\_started or s == '"':  
 current += s  
 if s == '"':  
 if not string\_started:  
 string\_started = True  
 else:  
 prev\_is\_operator = False  
 string\_started = False  
 string\_tokens.append(current)  
 current = ''  
 elif whitespace\_check(s) or s in brackets.keys() or s in brackets.values() or s == ',' or column == len(  
 lines[row]) - 1:  
 if current in var\_types:  
 var\_types\_tokens.append(current)  
 prev\_var\_type = current  
 if prev\_is\_operator:  
 prev\_is\_operator = False  
 errors.append(operator\_error(current, lines[row], row, column))  
 elif current in key\_words:  
 key\_word\_tokens.append(current)  
 if prev\_is\_operator and current != 'endl':  
 prev\_is\_operator = False  
 errors.append(operator\_error(current, lines[row], row, column))  
 elif current in operators:  
 operator\_tokens.append(current)  
 if prev\_is\_operator:  
 errors.append(operator\_error(current, lines[row], row, column))  
 prev\_is\_operator = True  
 elif is\_int(current):  
 integer\_tokens.append(current)  
 prev\_is\_operator = False  
 elif is\_float(current):  
 float\_tokens.append(current)  
 prev\_is\_operator = False  
 elif current in functions:  
 prev\_is\_operator = False  
 elif current in var\_tokens:  
 prev\_is\_operator = False  
 elif current != '':  
 prev\_is\_operator = False  
 if prev\_var\_type != '':  
 if current[0] == '\_' or current[0].isalpha():  
 var\_tokens[current] = prev\_var\_type  
 else:  
 errors.append(var\_name\_error(lines[row], row, column))  
  
 if column > len(lines[row]) - 1 and lines[row][column + 1:].strip()[0] != ',':  
 prev\_var\_type = ''  
 elif current != ':':  
 errors.append(lexical\_error(current, lines[row], row, column))  
 current = ''  
 else:  
 current += s  
  
 result = bracket\_check(lines)  
 if result:  
 errors.append(result)  
  
 token\_output('KEY WORDS', set(key\_word\_tokens))  
 token\_output('INTEGER CONSTANTS', integer\_tokens)  
 token\_output('FLOAT CONSTANTS', float\_tokens)  
 token\_output('STRING CONSTANTS', string\_tokens)  
 token\_output('OPERATORS', set(operator\_tokens))  
 token\_output('VARIABLE TYPES', set(var\_types\_tokens))  
 token\_output('VARIABLES', var\_tokens)  
 token\_output('ERRORS', errors)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()