

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Взаимно-рекурсивные функции и процедуры. Синтаксический**  
**анализатор понятия скобки**

Студент гр. 9384

\_\_\_\_\_

Гурин С.Н.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Разработать анализатор, проводящий синтаксический анализ строки с помощью взаимно-рекурсивных функций и процедур.

### **Задание.**

#### **ВАРИАНТ 6.**

Построить синтаксический анализатор для понятия *простое выражение*.

*простое\_выражение ::= простой\_идентификатор | (простое\_выражение  
знак\_операции простое\_выражение)*  
*простой\_идентификатор ::= буква*  
*знак\_операции ::= -/+/\**

### **Анализ задачи.**

Задача заключается в том, что нужно написать алгоритм, проверяющий правильность синтаксиса в введенном простом выражении. Его реализация происходит с помощью рекурсий.

### **Реализация.**

Была реализована программа, на вход которой подаются данные из файла input.txt, введенные пользователем. Программа проводит синтаксический анализ каждой строки из данного файла и создает файл output.txt, в котором записаны результаты проведенной работы. Так же результат проведенной работы выводится в консоль/терминал пользователя.

Проверка на синтаксис происходит с помощью рекурсивной функции `bool Check(string sentence){...}`. На вход данной функции подается строка, которую нужно анализировать. Далее происходит проверка является ли эта строка простым идентификатором или нет, если является, то функция возвращает значение `true`, если нет, то функция проверяет синтаксис простого выражения, проверяя первый и последний символ на наличие открытой и закрытой скобки. Затем происходит проверка на наличие простых выражений в первом простом

выражении, но перед этим происходит выполнение процедуры проверки на наличие соблюденной орфографии во всей строке. Если второй символ строки – “(“, то есть имеется наличие первого простого выражения в первом простом выражении, то происходит подсчет количества знаков, открытых и закрытых скобок. Так же происходит проверка на синтаксис скобок и знаков. После цикла подсчета количество закрытых и открытых скобок должно совпадать. Этот цикл проходит до конца первого главного простого выражение. Далее происходит считывание этого простого выражение в переменную `string a`. Затем происходит считывание второго простого выражение в переменную `string b`. После записи простых выражений в данные переменные происходит рекурсивный вызов функции `Check(a)`, `Check(b)`, благодаря которым происходит повторное выполнение алгоритма с выражениями в `a` и в `b`.

После выполнения алгоритма проверки синтаксиса строки, происходит запись результатов этой проверки в ранее созданный файл `output.txt`.

### Тестирование.

№ теста	input.txt	output.txt
1	<pre> 1 (a1+a) 2 (a+as) 3 (a+a) 4 (1+1) 5 aa 6 a 7 (aa) 8 (a) 9 a+a) 10 (a++a) 11 (+a+) 12 ((a*((a+s)-b))+((b*c)-(c+a))) 13 ((a*((a+s)-b))(+)+((b*c)-(c+a))) 14 ((a*((a+s)-b)))+(b*c)-(c+a))) 15 ((a*((a+s)-b))))+(b*c)-(c+a))) 16 (a+&amp;a) 17 ! 18 [a+c]</pre>	<pre> (a1+a) - False (a+as) - False (a+a) - True (1+1) - False aa - False a - True (aa) - False (a) - False a+a) - False (a++a) - False (+a+) - False ((a*((a+s)-b))+((b*c)-(c+a))) - True ((a*((a+s)-b))(+)+((b*c)-(c+a))) - False ((a*((a+s)-b)))+(b*c)-(c+a))) - False ((a*((a+s)-b))))+(b*c)-(c+a))) - False (a+&amp;a) - False ! - False [a+c] - False</pre>

## **Выводы.**

При выполнении данной лабораторной работы была реализована программа, которая считывает строки из отдельного файла, проверяет их на синтаксис, а затем записывает результат в созданный файл. Так же был изучен принцип работы рекурсивных функций, принцип работы со строкой и принцип работы с файлами на языке C++.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

bool Check_sin(string sentence, int n) {
    char signs[20] =
    {'!', '@', '#', '$', '%', '^', '&', '{', '}', '[', ']', '=', '?', '>', '<', ',', '.', '"',
    '\'', '~'};
    for (int i = 0; i < n - 2; i++) {
        if (isalpha(sentence[i]) && isalpha(sentence[i + 1])) {
            return false;
        }
        if (isdigit(sentence[i])) {
            return false;
        }
        for (int j = 0; j < 20; j++) {
            if (sentence[i] == signs[j]) {
                return false;
            }
        }
    }
}

bool Check(string sentence) {
    int n = sentence.length();
    string a = "", b = "";
    if (isalpha(sentence[0]) && n == 1) {
        return true;
    }
    else {
        if (sentence[0] == '(' && sentence[n - 1] == ')') {
            int count = 1;
            int count_of_open = 0;
```

```

int count_of_close = 0;
int signs = 0;
if (!Check_sin(sentence, n)) {
    return false;
}
if (sentence[1] == '(') {
    while (signs <= count_of_open && count != n-1) {
        if (sentence[count] == '(') {
            count_of_open++;
        }
        if (sentence[count] == ')') {
            count_of_close++;
        }
        if (sentence[count] == '+' || sentence[count] == '-'
|| sentence[count] == '*') {
            signs++;
        }
        if (sentence[count+1] == ')') && (sentence[count] ==
'+' || sentence[count] == '-' || sentence[count] == '*')) {
            return false;
        }
        if (sentence[count - 1] == '(' && (sentence[count] ==
'+' || sentence[count] == '-' || sentence[count] == '*')) {
            return false;
        }
        count++;
    }
    if (count_of_close != count_of_open) {
        return false;
    }
    for (int i = 1; i < count - 1; i++) {
        a += sentence[i];
    }
}
else {
    while (sentence[count] != '+' && sentence[count] != '-'
&& sentence[count] != '*') {
        a += sentence[count];
    }
}

```



```

        return 0;
    }
    if (!file2.is_open()) {
        cout << "Error opening file (output.txt)" << endl;
        return 0;
    }
    while (!file.eof()) {
        getline(file, sentence);
        if (sentence == "") {

        }
        else{
            if (Check(sentence)) {
                cout << "\t" << sentence << " - True" << endl;
                file2 << sentence << " - True\n";
            }
            else {
                cout << "\t" << sentence << " - False" << endl;
                file2 << sentence << " - False\n";
            }
        }
    }
    file.close();
    file2.close();
}

```

**Название файла: Makefile**

```

all:
    g++ LR1_AiSD.cpp && clear && ./a.out

```