

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: Рекурсивные алгоритмы

Студент гр. 9381

Преподаватель

Прибылов Н.А.

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить принципы рекурсии, реализовать рекурсивный алгоритм.

Задание.

Вариант 12

Построить синтаксический анализатор для понятия *скобки*.

скобки::=*квадратные* | *круглые* | *фигурные*

квадратные::=[*круглые* *фигурные*] | +

круглые::=(*фигурные* *квадратные*) | -

фигурные::={*квадратные* *круглые*} | 0

Описание алгоритма.

В основе синтаксического анализатора для понятия *скобки* лежит алгоритм, реализованный на трёх взаимно рекурсивных функциях, соответствующих понятиям *квадратные*, *круглые* и *фигурные*. Точкой входа в алгоритм является функция, соответствующая понятию *скобки*, она же по первому символу в проверяемой последовательности определяет, какую из трёх функций запустить первой (либо сразу вернуть отрицательный результат, если первый символ не является допустимым).

Проверка каждого из трёх рекурсивных понятий происходит схожим образом: проверяется первый символ на соответствие открывающей скобке, либо на терминальный символ («+», «-» или «0»). Во втором случае считается, что последовательность символов прошла проверку. В первом случае далее идёт проверка следующих двух подпоследовательностей символов на соответствие двум другим понятиям (например, для понятия *квадратные* проверяются понятия *круглые* и *фигурные*). В случае положительного результата проверки каждого из этих двух понятий проверяется следующий символ на соответствие закрывающей скобке. В случае успеха текущая последовательность считается прошедшей проверку. Если на каком-либо из этапов результат отрицательный, значит, последовательность символов не прошла проверку.

Описание функций.

analyzer(std::ifstream& infile, std::ofstream& outfile):

Считывает данные построчно, для каждой строки вызывая функцию *analyzeLine*, после чего выводит результат анализа текущей строки.

analyzeLine(const std::string& line, int& pos, std::ofstream& outfile, const int recLevel):

Анализирует текущую строку, запуская функцию *brackets*. Если строка не прошла проверку, возвращает результат работы функции *brackets*, если же *brackets* вернула положительный результат, проверяет наличие лишних символов на конце строки, и возвращает окончательный результат.

brackets(const std::string& line, int& pos, std::ofstream& outfile, const int recLevel):

Проверяет строку на соответствие понятию «скобки», проверяя первый символ в строке. Если первый символ в строке не является допустимым вариантом, возвращает отрицательный результат, иначе запускает одну из трёх функций: *square*, *round* или *curly* в соответствии с начальным символом.

square(const std::string& line, int& pos, std::ofstream& outfile, const int recLevel):

Проверяет подпоследовательность символов на соответствие понятию «квадратные»: если первый символ не является «+» или «[», то возвращает отрицательный результат. Если первый символ – это «+», то возвращает положительный результат, если же это «[», то далее идёт проверка следующих двух подпоследовательностей символов на соответствие понятиям «круглые» и «фигурные» с помощью функций *round* и *curly* соответственно. Если любая из этих функций вернула отрицательный результат, значит текущая последовательность также не соответствует понятию «квадратные» и функция

возвращает отрицательный результат. В противном случае, проверяется следующий символ: если он «]», то возвращается положительный результат, иначе отрицательный.

Функции *round* и *curly* ведут себя практически идентичным образом ввиду схожести понятий «квадратные», «круглые» и «фигурные».

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1	AAA	Проверяется: AAA Глубина рекурсии 0: A ЭТО НЕ СКОБКИ! Самый первый символ ошибочен.
2	+	Проверяется: + Глубина рекурсии 1: + ЭТО СКОБКИ!
3	(0+)	Проверяется: (0+) Глубина рекурсии 1: (Глубина рекурсии 2: (0 Глубина рекурсии 2: (0+ Глубина рекурсии 1: (0+) ЭТО СКОБКИ!
4	{+-	Проверяется: {+- Глубина рекурсии 1: { Глубина рекурсии 2: {+ Глубина рекурсии 2: {+- Глубина рекурсии 1: {+- ЭТО НЕ СКОБКИ! Ожидался символ '}'.

5	[({ + - } [- 0]) 0] 0 0	<p>Проверяется: [({ + - } [- 0]) 0] 0</p> <p>Глубина рекурсии 1: [</p> <p>Глубина рекурсии 2: [(</p> <p>Глубина рекурсии 3: [({</p> <p>Глубина рекурсии 4: [({ +</p> <p>Глубина рекурсии 4: [({ + -</p> <p>Глубина рекурсии 3: [({ + - }</p> <p>Глубина рекурсии 3: [({ + - } [</p> <p>Глубина рекурсии 4: [({ + - } [-</p> <p>Глубина рекурсии 4: [({ + - } [- 0</p> <p>Глубина рекурсии 3: [({ + - } [- 0]</p> <p>Глубина рекурсии 2: [({ + - } [- 0])</p> <p>Глубина рекурсии 2: [({ + - } [- 0]) 0</p> <p>Глубина рекурсии 1: [({ + - } [- 0]) 0]</p> <p>Глубина рекурсии 0: [({ + - } [- 0]) 0] 0</p> <p>ЭТО НЕ СКОБКИ!</p> <p>Встретился лишний символ.</p>
6	[({ + - } [- 0]) 0]	<p>Проверяется: [({ + - } [- 0]) 0]</p> <p>Глубина рекурсии 1: [</p> <p>Глубина рекурсии 2: [(</p> <p>Глубина рекурсии 3: [({</p> <p>Глубина рекурсии 4: [({ +</p> <p>Глубина рекурсии 4: [({ + -</p> <p>Глубина рекурсии 3: [({ + - }</p> <p>Глубина рекурсии 3: [({ + - } [</p> <p>Глубина рекурсии 4: [({ + - } [-</p> <p>Глубина рекурсии 4: [({ + - } [- 0</p> <p>Глубина рекурсии 3: [({ + - } [- 0]</p> <p>Глубина рекурсии 2: [({ + - } [- 0])</p> <p>Глубина рекурсии 2: [({ + - } [- 0]) 0</p> <p>Глубина рекурсии 1: [({ + - } [- 0]) 0]</p> <p>ЭТО СКОБКИ!</p>
7	[({ + - } [- 0] -) 0]	<p>Проверяется: [({ + - } [- 0] -) 0]</p> <p>Глубина рекурсии 1: [</p>

		<p>Глубина рекурсии 2: [(</p> <p>Глубина рекурсии 3: [({</p> <p>Глубина рекурсии 4: [({ +</p> <p>Глубина рекурсии 4: [({ + -</p> <p>Глубина рекурсии 3: [({ + - }</p> <p>Глубина рекурсии 3: [({ + - } [</p> <p>Глубина рекурсии 4: [({ + - } [-</p> <p>Глубина рекурсии 4: [({ + - } [- 0</p> <p>Глубина рекурсии 3: [({ + - } [- 0]</p> <p>Глубина рекурсии 2: [({ + - } [- 0] -</p> <p>ЭТО НЕ СКОБКИ!</p> <p>Ожидался символ ') '.</p>
8	[({ + - } [- 0 +]) 0]	<p>Проверяется: [({ + - } [- 0 +]) 0]</p> <p>Глубина рекурсии 1: [</p> <p>Глубина рекурсии 2: [(</p> <p>Глубина рекурсии 3: [({</p> <p>Глубина рекурсии 4: [({ +</p> <p>Глубина рекурсии 4: [({ + -</p> <p>Глубина рекурсии 3: [({ + - }</p> <p>Глубина рекурсии 3: [({ + - } [</p> <p>Глубина рекурсии 4: [({ + - } [-</p> <p>Глубина рекурсии 4: [({ + - } [- 0</p> <p>Глубина рекурсии 3: [({ + - } [- 0 +</p> <p>ЭТО НЕ СКОБКИ!</p> <p>Ожидался символ '] '.</p>
9	[({ + - } [0 - 0]) 0]	<p>Проверяется: [({ + - } [0 - 0]) 0]</p> <p>Глубина рекурсии 1: [</p> <p>Глубина рекурсии 2: [(</p> <p>Глубина рекурсии 3: [({</p> <p>Глубина рекурсии 4: [({ +</p> <p>Глубина рекурсии 4: [({ + -</p> <p>Глубина рекурсии 3: [({ + - }</p> <p>Глубина рекурсии 3: [({ + - } [</p> <p>Глубина рекурсии 4: [({ + - } [0</p>

		<p>ЭТО НЕ СКОБКИ!</p> <p>Ожидался символ '(' или '-'.</p>
10	<code>[({ + - } a [- 0]) 0]</code>	<p>Проверяется: <code>[({ + - } a [- 0]) 0]</code></p> <p>Глубина рекурсии 1: <code>[</code></p> <p>Глубина рекурсии 2: <code>[(</code></p> <p>Глубина рекурсии 3: <code>[({</code></p> <p>Глубина рекурсии 4: <code>[({ +</code></p> <p>Глубина рекурсии 4: <code>[({ + -</code></p> <p>Глубина рекурсии 3: <code>[({ + - }</code></p> <p>Глубина рекурсии 3: <code>[({ + - } a</code></p> <p>ЭТО НЕ СКОБКИ!</p> <p>Ожидался символ '[' или '+'.</p>
11	<code>({ + - } [() { }])</code>	<p>Проверяется: <code>({ + - } [() { }])</code></p> <p>Глубина рекурсии 1: <code>(</code></p> <p>Глубина рекурсии 2: <code>({</code></p> <p>Глубина рекурсии 3: <code>({ +</code></p> <p>Глубина рекурсии 3: <code>({ + -</code></p> <p>Глубина рекурсии 2: <code>({ + - }</code></p> <p>Глубина рекурсии 2: <code>({ + - } [</code></p> <p>Глубина рекурсии 3: <code>({ + - } [(</code></p> <p>Глубина рекурсии 4: <code>({ + - } [()</code></p> <p>ЭТО НЕ СКОБКИ!</p> <p>Ожидался символ '[' или '0'.</p>
13	<code>[({ + - } [- 0]) { [- 0] (0 +) }]</code>	<p>Проверяется: <code>[({ + - } [- 0]) { [- 0] (0 +) }]</code></p> <p>Глубина рекурсии 1: <code>[</code></p> <p>Глубина рекурсии 2: <code>[(</code></p> <p>Глубина рекурсии 3: <code>[({</code></p> <p>Глубина рекурсии 4: <code>[({ +</code></p> <p>Глубина рекурсии 4: <code>[({ + -</code></p> <p>Глубина рекурсии 3: <code>[({ + - }</code></p> <p>Глубина рекурсии 3: <code>[({ + - } [</code></p> <p>Глубина рекурсии 4: <code>[({ + - } [-</code></p> <p>Глубина рекурсии 4: <code>[({ + - } [- 0</code></p>

		<p>Глубина рекурсии 3: [(+-)[-0]</p> <p>Глубина рекурсии 2: [(+-)[-0])</p> <p>Глубина рекурсии 2: [(+-)[-0)]{</p> <p>Глубина рекурсии 3: [(+-)[-0)]{[</p> <p>Глубина рекурсии 4: [(+-)[-0)]{[-</p> <p>Глубина рекурсии 4: [(+-)[-0)]{[-0</p> <p>Глубина рекурсии 3: [(+-)[-0)]{[-0]</p> <p>Глубина рекурсии 3: [(+-)[-0)]{[-0](</p> <p>Глубина рекурсии 4: [(+-)[-0)]{[-0](0</p> <p>Глубина рекурсии 4: [(+-)[-0)]{[-0](0+</p> <p>Глубина рекурсии 3: [(+-)[-0)]{[-0](0+)</p> <p>Глубина рекурсии 2: [(+-)[-0)]{[-0](0+)}]</p> <p>Глубина рекурсии 1: [(+-)[-0)]{[-0](0+)}]]</p> <p>ЭТО СКОБКИ!</p>
14	(0[(0+)0])	<p>Проверяется: (0[(0+)0])</p> <p>Глубина рекурсии 1: (</p> <p>Глубина рекурсии 2: (0</p> <p>Глубина рекурсии 2: (0[</p> <p>Глубина рекурсии 3: (0[(</p> <p>Глубина рекурсии 4: (0[(0</p> <p>Глубина рекурсии 4: (0[(0+</p> <p>Глубина рекурсии 3: (0[(0+)</p> <p>Глубина рекурсии 3: (0[(0+)0</p> <p>Глубина рекурсии 2: (0[(0+)0]</p> <p>Глубина рекурсии 1: (0[(0+)0])</p> <p>ЭТО СКОБКИ!</p>
15	{[-0](+-)[-+-]}}	<p>Проверяется: {[-0](+-)[-+-]}}</p> <p>Глубина рекурсии 1: {</p> <p>Глубина рекурсии 2: {[</p> <p>Глубина рекурсии 3: {[-</p> <p>Глубина рекурсии 3: {[-0</p> <p>Глубина рекурсии 2: {[-0]</p> <p>Глубина рекурсии 2: {[-0](</p> <p>Глубина рекурсии 3: {[-0]({</p>

		Глубина рекурсии 4: $\{[-0]({+}$ Глубина рекурсии 4: $\{[-0]({+-}$ Глубина рекурсии 3: $\{[-0]({+-}$ Глубина рекурсии 3: $\{[-0]({+-}[$ Глубина рекурсии 4: $\{[-0]({+-}[-$ Глубина рекурсии 4: $\{[-0]({+-}[-{$ Глубина рекурсии 5: $\{[-0]({+-}[-{+$ Глубина рекурсии 5: $\{[-0]({+-}[-{+-}$ Глубина рекурсии 4: $\{[-0]({+-}[-{+-}$ Глубина рекурсии 3: $\{[-0]({+-}[-{+-}$ Глубина рекурсии 2: $\{[-0]({+-}[-{+-})$ Глубина рекурсии 1: $\{[-0]({+-}[-{+-}))$ ЭТО СКОБКИ!
--	--	---

Выводы.

Были изучены принципы рекурсии, был построен синтаксический анализатор для определённого понятия с использованием рекурсивных алгоритмов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

char TASK[] = "Синтаксический анализатор понятия скобки:\n"
              "скобки ::= квадратные | круглые | фигурные\n"
              "квадратные ::= [круглые фигурные] | +\n"
              "круглые ::= (фигурные квадратные) | -\n"
              "фигурные ::= {квадратные круглые} | 0\n\n";

// Классификация результата анализатора
enum class Result {
    GOOD, BADVERYFIRSTCHAR,
    BADFIRSTCHARSQUARE, BADLASTCHARSQUARE,
    BADFIRSTCHARROUND, BADLASTCHARROUND,
    BADFIRSTCHARCURLY, BADLASTCHARCURLY,
    EXCESSCHAR
};

// Анализатор нескольких последовательностей символов,
// принимает открытые файловые потоки:
// infile для чтения строк и outfile для записи результатов
void analyzer(std::ifstream& infile, std::ofstream& outfile);

// Анализирует последовательность символов,
// принимает строку line для проверки,
// позицию pos символа, с которого начинается проверка в строке,
// и уровень глубины рекурсии recLevel
Result analyzeLine(const std::string& line, int& pos,
std::ofstream& outfile, const int recLevel = 1);

// Проверяет последовательность символов на соответствие понятию
"скобки"
Result brackets(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel);

// Следующие три функции проверяют последовательность символов на
соответствие понятиям
// "квадратные", "круглые" и "фигурные" соответственно
Result square(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel);
Result round(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel);
Result curly(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel);

// Выводит первые pos символов из line на консоль
// и в файл, связанный с потоком outfile,
// а так же глубину рекурсии recLevel
void printLog(const std::string& line, const int pos,
std::ofstream& outfile, const int recLevel);
```

```

void analyzer(std::ifstream& infile, std::ofstream& outfile)
{
    std::string line;
    while(getline(infile, line)) { // читает строки, пока
не конец файла,
        if (!line.length()) continue; // пропускает пустые
        std::cout << "Проверяется: " << line << "\n";
        outfile << "Проверяется: " << line << "\n";
        int pos = 0;
        Result k = analyzeLine(line, pos, outfile, 1);

        switch (k) { //для обработки результата (верного и
ошибочного)
            case Result::GOOD:
                std::cout << "ЭТО СКОБКИ!\n\n";
                outfile << "ЭТО СКОБКИ!\n\n";
                break;
            case Result::BADVERYFIRSTCHAR:
                std::cout << "ЭТО НЕ СКОБКИ!\nСамый первый символ
ошибочен.\n\n";
                outfile << "ЭТО НЕ СКОБКИ!\nСамый первый символ
ошибочен.\n\n";
                break;
            case Result::BADFIRSTCHARSQUARE:
                std::cout << "ЭТО НЕ СКОБКИ!\nОжидался символ '['\n'
или \'+\'.\n\n";
                outfile << "ЭТО НЕ СКОБКИ!\nОжидался символ '['\n'
или \'+\'.\n\n";
                break;
            case Result::BADLASTCHARSQUARE:
                std::cout << "ЭТО НЕ СКОБКИ!\nОжидался
символ ']\n'.\n\n";
                outfile << "ЭТО НЕ СКОБКИ!\nОжидался
символ ']\n'.\n\n";
                break;
            case Result::BADFIRSTCHARROUND:
                std::cout << "ЭТО НЕ СКОБКИ!\nОжидался символ '('\n'
или \'-\'.\n\n";
                outfile << "ЭТО НЕ СКОБКИ!\nОжидался символ '('\n'
или \'-\'.\n\n";
                break;
            case Result::BADLASTCHARROUND:
                std::cout << "ЭТО НЕ СКОБКИ!\nОжидался
символ ')\n'.\n\n";
                outfile << "ЭТО НЕ СКОБКИ!\nОжидался
символ ')\n'.\n\n";
                break;
            case Result::BADFIRSTCHARCURLY:
                std::cout << "ЭТО НЕ СКОБКИ!\nОжидался символ '{\n'
или '\0\'.\n\n";
                outfile << "ЭТО НЕ СКОБКИ!\nОжидался символ '{\n'
или '\0\'.\n\n";
                break;
            case Result::BADLASTCHARCURLY:
                std::cout << "ЭТО НЕ СКОБКИ!\nОжидался
символ '}\n'.\n\n";

```

```

                                outfile    << "ЭТО НЕ СКОБКИ!\nОжидался
символ \'}\'.\n\n";
                                break;
                                case Result::EXCESSCHAR:
                                    std::cout << "ЭТО НЕ СКОБКИ!\nВстретился лишний
символ.\n\n";
                                    outfile    << "ЭТО НЕ СКОБКИ!\nВстретился лишний
символ.\n\n";
                                    break;
                                default:
                                    std::cout << "Неизвестная ошибка.\n\n";
                                    outfile    << "Неизвестная ошибка.\n\n";
                                }
                                line.clear();
                            }
                        }

Result analyzeLine(const std::string& line, int& pos,
std::ofstream& outfile, const int recLevel)
{
    Result k = brackets(line, pos, outfile, recLevel);
    if (k == Result::GOOD && pos != line.length()) { // проверка на
отсутствие лишних символов на конце
        printLog(line, ++pos, outfile, 0);
        return Result::EXCESSCHAR;
    }
    return k;
}

Result brackets(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel)
{
    char c; Result k;
    c = line[pos];
    if (c == '[' || c == '+') {
        k = square(line, pos, outfile, recLevel);
    } else if (c == '(' || c == '-') {
        k = round(line, pos, outfile, recLevel);
    } else if (c == '{' || c == '0') {
        k = curly(line, pos, outfile, recLevel);
    } else {
        printLog(line, ++pos, outfile, 0);
        k = Result::BADVERYFIRSTCHAR;
    }
    return k;
}

Result square(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel)
{
    char c; Result k;
    c = line[pos++];
    printLog(line, pos, outfile, recLevel);

    if (c == '+') {
        return Result::GOOD;
    }
    if (c != '[') {

```

```

        return Result::BADFIRSTCHAR SQUARE;
    }

    k = round(line, pos, outfile, recLevel + 1);
    if (k != Result::GOOD) {
        return k;
    }
    k = curly(line, pos, outfile, recLevel + 1);
    if (k != Result::GOOD) {
        return k;
    }

    c = line[pos++];
    printLog(line, pos, outfile, recLevel);
    if (c != ']') {
        return Result::BADLASTCHAR SQUARE;
    }

    return Result::GOOD;
}

Result round(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel)
{
    char c; Result k;
    c = line[pos++];
    printLog(line, pos, outfile, recLevel);

    if (c == '-') {
        return Result::GOOD;
    }
    if (c != '(') {
        return Result::BADFIRSTCHAR ROUND;
    }

    k = curly(line, pos, outfile, recLevel + 1);
    if (k != Result::GOOD) {
        return k;
    }
    k = square(line, pos, outfile, recLevel + 1);
    if (k != Result::GOOD) {
        return k;
    }

    c = line[pos++];
    printLog(line, pos, outfile, recLevel);
    if (c != ')') {
        return Result::BADLASTCHAR ROUND;
    }

    return Result::GOOD;
}

Result curly(const std::string& line, int& pos, std::ofstream&
outfile, const int recLevel)
{
    char c; Result k;
    c = line[pos++];

```

```

    printLog(line, pos, outfile, recLevel);

    if (c == '0') {
        return Result::GOOD;
    }
    if (c != '{') {
        return Result::BADFIRSTCHARCURLY;
    }

    k = square(line, pos, outfile, recLevel + 1);
    if (k != Result::GOOD) {
        return k;
    }
    k = round(line, pos, outfile, recLevel + 1);
    if (k != Result::GOOD) {
        return k;
    }

    c = line[pos++];
    printLog(line, pos, outfile, recLevel);
    if (c != '}') {
        return Result::BADLASTCHARCURLY;
    }

    return Result::GOOD;
}

```

```

void printLog(const std::string& line, const int pos,
std::ofstream& outfile, const int recLevel)
{
    std::cout << std::string(recLevel, '\t') << "Глубина рекурсии "
<< std::setw(2) << recLevel << ": " << line.substr(0, pos) << "\n";
    outfile << std::string(recLevel, '\t') << "Глубина рекурсии "
<< std::setw(2) << recLevel << ": " << line.substr(0, pos) << "\n";
}

```

```

int main()
{
    std::string readFileName, logFileName;
    std::ifstream infile; // поток для чтения
    std::ofstream outfile; // поток для вывода

    std::cout << TASK; // вывод задания на экран
    do {
        // проверка на корректность введенного имени
        // файла
        std::cout << "Введите название файла для считывания данных:
";
        getline(std::cin, readFileName);
        infile.open(readFileName);
        if (!infile) {
            std::cout << "Файла \"" << readFileName << "\" не
существует.\n";
        }
    } while (!infile);

    std::cout << "Введите название файла для записи промежуточных
результатов: ";
}

```

```
    getline(std::cin, logFileName);  
    outfile.open(logFileName);  
  
    analyzer(infile, outfile);  
  
    infile.close();  
    outfile.close();  
    return 0;  
}
```