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 _____ Прибылов Н.А.

Преподаватель _____ 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::istream& 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]00	Проверяется: [({+-}[-0])0]00 Глубина рекурсии 1: [Глубина рекурсии 2: [(Глубина рекурсии 3: [({ Глубина рекурсии 4: [({+ Глубина рекурсии 4: [({+- Глубина рекурсии 3: [({+-} Глубина рекурсии 3: [({+-}[Глубина рекурсии 4: [({+-}[- Глубина рекурсии 4: [({+-}[-0 Глубина рекурсии 3: [({+-}[-0] Глубина рекурсии 2: [({+-}[-0]) Глубина рекурсии 2: [({+-}[-0])0 Глубина рекурсии 1: [({+-}[-0])0] Глубина рекурсии 0: [({+-}[-0])0]0 ЭТО НЕ СКОБКИ! Встретился лишний символ.
6	[({+-}[-0])0]	Проверяется: [({+-}[-0])0] Глубина рекурсии 1: [Глубина рекурсии 2: [(Глубина рекурсии 3: [({ Глубина рекурсии 4: [({+ Глубина рекурсии 4: [({+- Глубина рекурсии 3: [({+-} Глубина рекурсии 3: [({+-}[Глубина рекурсии 4: [({+-}[- Глубина рекурсии 4: [({+-}[-0 Глубина рекурсии 3: [({+-}[-0] Глубина рекурсии 2: [({+-}[-0]) Глубина рекурсии 2: [({+-}[-0])0 Глубина рекурсии 1: [({+-}[-0])0] ЭТО СКОБКИ!
7	[({+-}[-0]-)0]	Проверяется: [({+-}[-0]-)0] Глубина рекурсии 1: [

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

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

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

15	{[-0]({+-}[-{+-}])}	Проверяется: {[-0]({+-}[-{+-}])}
		Глубина рекурсии 1: {
		Глубина рекурсии 2: {[
		Глубина рекурсии 3: {[-
		Глубина рекурсии 3: {[-0
		Глубина рекурсии 2: {[-0]
		Глубина рекурсии 2: {[-0](
		Глубина рекурсии 3: {[-0]({

		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>

const 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";
const char STOP[] = "STOP";

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

// Анализатор нескольких последовательностей символов,
// принимает поток infile (файловый или стандартный) для чтения строк
// и файловый поток outfile для записи результатов
void analyzer(std::istream& 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::istream& infile, std::ofstream& outfile)
    {
        std::string line;
        while(getline(infile, line)) {
            if (line == STOP) return;
            if (!line.length()) continue;
            std::cout << "Проверяется: " << line << "\n";
            outfile   << "Проверяется: " << line << "\n";
            int pos = 0;
            Result k = analyzeLine(line, pos, outfile, 1);

            switch (k) {
                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 \'[\'
или \'+\'.\n\n";
                    outfile   << "ЭТО НЕ СКОБКИ!\nОжидался символ \'[\'
или \'+\'.\n\n";
                    break;
                case Result::BADLASTCHARSQUARE:
                              std::cout  <<  "ЭТО  НЕ  СКОБКИ!\nОжидался
символ \']\'.\n\n";
                             outfile    <<  "ЭТО  НЕ  СКОБКИ!\nОжидался
символ \']\'.\n\n";
                    break;
                case Result::BADFIRSTCHARROUND:
                    std::cout << "ЭТО НЕ СКОБКИ!\nОжидался символ \'(\'
или \'-\'.\n\n";
                    outfile   << "ЭТО НЕ СКОБКИ!\nОжидался символ \'(\'
или \'-\'.\n\n";
                    break;
                case Result::BADLASTCHARROUND:
                              std::cout  <<  "ЭТО  НЕ  СКОБКИ!\nОжидался
символ \')\'.\n\n";
                             outfile    <<  "ЭТО  НЕ  СКОБКИ!\nОжидался
символ \')\'.\n\n";
                    break;
                case Result::BADFIRSTCHARCURLY:
                    std::cout << "ЭТО НЕ СКОБКИ!\nОжидался символ \'{\'
или \'0\'.\n\n";
                    outfile   << "ЭТО НЕ СКОБКИ!\nОжидался символ \'{\'
или \'0\'.\n\n";
                    break;
                case Result::BADLASTCHARCURLY:
```

```
                                std::cout  <<  "ЭТО  НЕ  СКОБКИ!\nОжидался
символ \'}\'.\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()) {
            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::BADFIRSTCHARSQUARE;
        }

           // проверка подпоследовательности на соответствие понятию
"круглые"
        k = round(line, pos, outfile, recLevel + 1);
        if (k != Result::GOOD) {
            return k;
        }
           // проверка подпоследовательности на соответствие понятию
"фигурные"
        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::BADLASTCHARSQUARE;
        }

        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::BADFIRSTCHARROUND;
        }

           // проверка подпоследовательности на соответствие понятию
"фигурные"
        k = curly(line, pos, outfile, recLevel + 1);
        if (k != Result::GOOD) {
            return k;
        }
           // проверка подпоследовательности на соответствие понятию
"квадратные"
        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::BADLASTCHARROUND;
        }
```

```
        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;
        }

           // проверка подпоследовательности на соответствие понятию
"квадратные"
        k = square(line, pos, outfile, recLevel + 1);
        if (k != Result::GOOD) {
            return k;
        }
           // проверка подпоследовательности на соответствие понятию
"круглые"
        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 inputFileName, logFileName;
        std::ifstream infile;
        std::ofstream outfile;

        std::cout << TASK;
        do {
```

```
                    std::cout << "Для считывания данных с клавиатуры
введите \"NUL\".\n"
                                    "Для считывания данных с файла введите
название файла: ";
                std::cin >> inputFileName;
                if (inputFileName == "NUL") break;
                infile.open(inputFileName);
                if (!infile) {
                        std::cout << "Файла \"" << inputFileName << "\" не
существует.\n";
                }
            } while (!infile);

             std::cout << "Введите название файла для записи промежуточных
результатов: ";
            std::cin >> logFileName;
            outfile.open(logFileName);

            std::cout << "\nЧтение данных прекратится на строке \"" << STOP
<< "\".\n";
            if (inputFileName == "NUL") {
                std::cout << "Вводите данные:\n";
                analyzer(std::cin, outfile);
            } else {
                analyzer(infile, outfile);
            }

            if (infile.is_open()) {
                infile.close();
            }
            outfile.close();
            return 0;
        }
```