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

Выполнение работы.

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]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] 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>

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,
    excessCharacter
};

// 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;       // пропуская пустые
              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::excessCharacter:
                       std::cout << "ЭТО НЕ СКОБКИ!\nВстретился лишний
символ.\n\n";
                      outfile   << "ЭТО НЕ СКОБКИ!\nВстретился лишний
символ.\n\n";
                    break;
                case Result::badVeryFirstChar:
                    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::excessCharacter;
        }
        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;
        }

        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::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;
        }

        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::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;
        }

        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 << "Глубина рекурсии " << std::setw(2) << recLevel <<
": " << line.substr(0, pos) << "\n";
        outfile  << "Глубина рекурсии " << std::setw(2) << recLevel <<
": " << line.substr(0, pos) << "\n";
    }

    int main()
    {
        std::string readFileName, logFileName;
        std::ifstream infile;
        std::ofstream outfile;

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