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

# ОТЧЕТ

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Рекурсия**

Студентка гр. 8304 _____ Чечеткина К.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

**Цель работы.**

Изучить основы рекурсии и составления эффективных алгоритмов.

**Постановка задачи.**

1) Разработать программу, использующую рекурсию;
2) Сопоставить рекурсивное и итеративное решение задачи;
3) Сделать вывод о целесообразности и эффективности рекурсивного подхода для решения данной задачи

Вариант 20.

Построить синтаксический анализатор понятия *список_параметров*.

*список_параметров::=параметр | параметр, список_параметров*

*параметр::=имя=цифра цифра | имя=( список_параметров)*

*имя::=буква буква буква*

**Описание алгоритма.**

Для решения поставленной задачи была реализована рекурсивная функция recursion, которая анализирует строку на вхождение в нее параметра, выделяет его из строки и рекурсивно вызывает саму себя. Сначала функция проверяет соответствует ли отрезок в 6 символов простейшему условию *список_параметров::=буква буква буква=цифра цифра.* Далее функция проверяет более сложные условия с вложенными простейшими. При нахождении второго рекурсивно обращается  сама к себе, сдвигая границы отрезка. Функция работает, пока правая и левая границы не совпадут или не найдется отрезок не соответствующий условию.

**Спецификация программы.**

Программа предназначена для синтаксического анализа выражения методом рекурсии. Программа написана на языке C++. Входными данными

является строка. Выходными данными являются обрабатываемые отрезки и конечное подтверждение выполнения задачи.

**Описание функций.**

1) Функция read(char *).

Функция считывает строку и считает ее длину.

2) Функция test(int ,int ).

Вывод обрабатываемые отрезки строки.

3)Функция Recursion(int, int).

Основная функция программы.

4)Функция write(std::string )

Вывод результата работы программы

**Вывод.**

Был получен опыт работы с рекурсией и с построением синтаксического анализатора. На мой взгляд, итеративное решение поставленной задачи более эффективно.

## ПРИЛОЖЕНИЕ

## 1. ТЕСТИРОВАНИЕ:

Работа программы для строки fgt=(qwe=12,hot=(cat=34,bal=77)

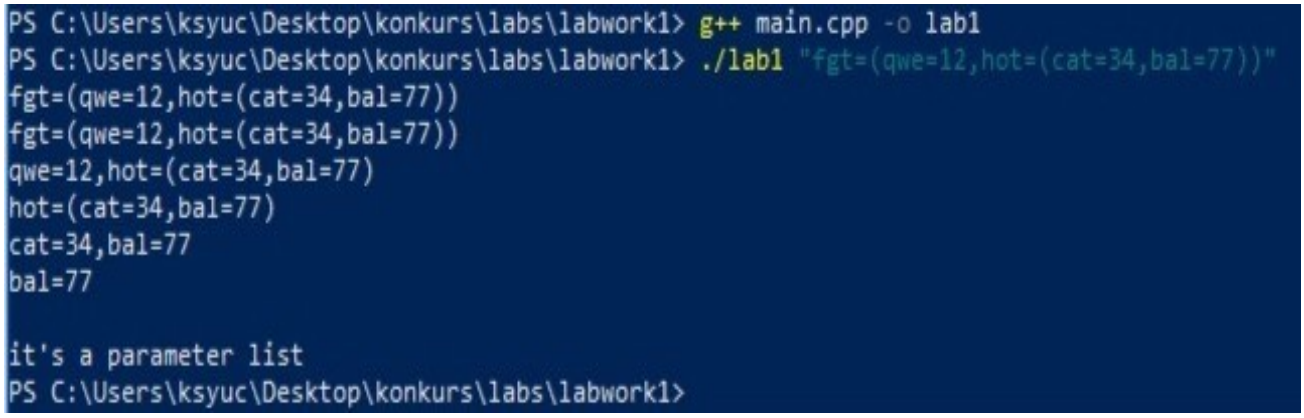


Таблица ввода/вывода тестирования программы

| Входная строка | Вывод программы |
|---|---|
| are=12 | it's a parameter list |
| are=123 | it's not a parameter list |
| art=66,qwe=77 | it's a parameter list |
| solt=45 | it's not a parameter list |
| are=(two=45,one=34) | it's a parameter list |
| are=(two=45,one=344) | it's not a parameter list |
| are=(twooo=45,one=34) | it's not a parameter list |
| are=22,rer=(sre=67,ftr=88) | it's a parameter list |
| are=22,rer=(sre=67,ftr) | it's not a parameter list |
| are=(are=12,tre=34),res=54 | it's not a parameter list |
| wet=78,ret=(day=66,ree=(tur=33,les=(qwe=09,jet=78))) | it's a parameter list |

## 2. ИСХОДНЫЙ КОД:

```
#include <iostream>
#include <string>
#include <fstream>
#include <conio.h>

class MyRecursion{
private:
    std::string str;
public:
    int length;
```

```
    int recursion(int, int);
    void read(char* );
    void testWorkingDistance(int, int);
    void write(std::string);
MyRecursion();
MyRecursion(std::string&);
};

void MyRecursion::write(std::string result){
    std::cout << result << std::endl;
}

void MyRecursion::read(char* text) {
    str = text;
    length = str.length()-1;
}


void MyRecursion::testWorkingDistance(int leftBorder,int rightBorder){
    for(int i = leftBorder; i <= rightBorder; i++)
        std::cout << str[i];
    std::cout << std::endl;

}

MyRecursion::MyRecursion(std::string &string) {
    this->str = string;
    this->length = str.length()-1;
}

MyRecursion::MyRecursion() {
    this->str = "";
    this->length = 0;
}

int MyRecursion::recursion(int leftBorder, int rightBorder) {

    testWorkingDistance(leftBorder, rightBorder);

    if (leftBorder >= rightBorder)
    {
        write("it's a parameter list");
        return 0;
    }

    bool verification = true;
    int symbolCount = 0;

    for (symbolCount; symbolCount < 3; symbolCount++) {
        if (!isalpha(str[leftBorder + symbolCount])) {
            verification = false;
        }
    }

    if (str[leftBorder + symbolCount] != '=') {
        verification = false;
    }
    symbolCount++;

    for (symbolCount; symbolCount < 6; symbolCount++) {
        if (!isalnum(str[leftBorder + symbolCount])) {
```

```
            verification = false;
        }
    }

    if (verification){
        if ((isalnum(str[leftBorder + 6])!= 0 || isalpha(str[leftBorder + 6] !=
0))){
            write("it's not a parameter list");
            return 0;
        }
        recursion(leftBorder + ((str[leftBorder + 6] = ',') ? 7:6),
rightBorder);
        return 0;
    }

    verification = true;
    symbolCount = 0;

    for (symbolCount; symbolCount < 3; symbolCount++) {
        if (!isalpha(str[leftBorder + symbolCount])) {
            verification = false;
        }
    }

    bool notEqualSign = str[leftBorder + 3] != '=';
    bool notLeftBracket = str[leftBorder + 4] != '(';
    bool notRightBracket = str[rightBorder] != ')';

    if ((!verification) || notEqualSign || notLeftBracket || notRightBracket){
        verification = false;
    }

    if (verification){
        recursion(leftBorder + 5,rightBorder - 1);
    } else {
        write("it's not a parameter list");
    }
    return 0;
}



int main(int argc, char* argv[]) {

    if (argc == 1){
        std::cout << "argc: " << argc << std::endl;
        std::cout << "argv[0]: " << argv[0] << std::endl;
       // std::cout << "argv[1]:" << argv[1] << std::endl;
        std::cout << "enter" << std::endl;
        std::string line;
        std::ifstream in;
        in.open("D:\\study\\test.txt"); //открываем файл
        if (in.is_open())
        {
            std::cout << "file opened" << std::endl;
            while (getline(in, line))
            {
                MyRecursion Text(line);
                Text.recursion(0, Text.length);
                std::cout<<std::endl;
            }
```

```
        }
        in.close();//закрываем файл
    }else{
        //std::cout << argv[1] << std::endl;
        MyRecursion Text;//(argv[1]);
        Text.read(argv[1]);
        Text.recursion(0, Text.length);

    }
    return 0;
}
```