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: Рекурсия

Студент гр. 9304 _____ Ламбин А.В.

Преподаватель _____ Фиалковский М.С.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++.

Задание.

Вариант — 23.

Разработать программу, которая, имея на входе заданное логическое выражение, не содержащее вхождений идентификаторов, вычисляет значение этого выражения и печатает само выражение и его значение.

Выполнение работы.

На вход программе подаётся строка, являющаяся логическим выражением.

В начале программы объявляется переменная *str* типа *string*. В случае, если количество аргументов командной строки не превышает одного (сам вызов программы), то в переменную *str* записывается строка из стандартного потока ввода. В противном случае – записывается строка, идущая следующим аргументом (это сделано для более простого тестирования программы). Затем с помощью регулярных выражений убираются лишние пробелы, а с помощью функции *transform()* из библиотеки *<algorithm>* все буквы приводятся к верхнему регистру. В конце функции *main()* выводятся строка *str* и результат работы функции *analysis(str)*.

В начале функции *analysis()* инициализируются четыре переменные, используемые для хранения индексов. В случае, если в строке имеется хотя бы один знак *‘(’*, то ищется соответствующий знак *‘)’*, после чего вся подстрока, начиная от *‘(’* и заканчивая *‘)’*, заменяется на результат работы функции *analysis()* от подстроки, находящейся в этих скобках. Если скобок в строке нет, то ищется подстрока *“NOT ”*, после чего подстрока *“NOT [логическое_выражение]”* заменяется на *“TRUE”*, если логическое выражение было *“FALSE”*, или на *“FALSE”*, если оно было *“TRUE”*. Для простоты реализации была написана

функция *converter()*, возвращающая *true*, если входная строка *“TRUE”*, или *false* в противном случае. Аналогично программа работает, когда ищет *“ AND ”* и *“ OR ”*. В конце функция возвращает значение *str*.

Разработанный программный код см. в приложении А.

Тестирование.

Запуск программы начинается с ввода команды *“make”*, что приведёт к компиляции программы и созданию исполняемого файла *lab1*. Запуск программы производится командой *“./lab1”* и последующим вводом строки, содержащей логическое выражение.

Тестирование производится с помощью скрипта *script.py*. Запуск скрипта производится командой *“python3 script.py”* в директории *tests*.

Результаты тестирования см. в приложении Б.

Выводы.

Было проведено ознакомление с основными понятиями и приёмами рекурсивного программирования, были получены навыки программирования рекурсивных процедур и функций на языке программирования C++.

Разработана программа, рекурсивно вычисляющая значение входного логического выражения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include <regex>

bool converter (const std::string &str) {
    if (str == "TRUE")
        return true;
    return false;
}

std::string analysis (std::string str) {
    unsigned int pos1 = 0, pos2 = 0;
    unsigned int pos0 = 0, pos3 = 0;

    if (str.find('(') != std::string::npos) {
        int n = 0;
        for (int i = 0; i < str.size(); i++) {
            if (str[i] == '(') {
                if (n == 0) {
                    pos1 = i;
                }
                n++;
            } else if (str[i] == ')') {
                n--;
                if (n == 0) {
                    pos2 = i;
                    str.replace(pos1, pos2 - pos1 + 1,
analysis(str.substr(pos1 + 1, pos2 - pos1 - 1)));
                    str = analysis(str);
                }
            }
        }
    } else if (str.find("NOT ") != std::string::npos) {
        pos1 = str.find("NOT ");
        pos2 = pos1 + 3;
        if (str.find(' ', pos2 + 1) != std::string::npos)
            pos3 = str.find(' ', pos2 + 1) - 1;
        else
            pos3 = str.size() - 1;

        if (!converter(str.substr(pos2 + 1, pos3 - pos2)))
            str.replace(pos1, pos3 - pos1 + 1, "TRUE");
        else
            str.replace(pos1, pos3 - pos1 + 1, "FALSE");
        str = analysis(str);
    } else if (str.find(" AND ") != std::string::npos) {
        pos1 = str.find(" AND ");
        pos2 = pos1 + 4;
        if (str.rfind(' ', pos1 - 1) != std::string::npos)
            pos0 = str.rfind(' ', pos1 - 1) + 1;
```

```
            else
                pos0 = 0;
            if (str.find(' ', pos2 + 1) != std::string::npos)
                pos3 = str.find(' ', pos2 + 1) - 1;
            else
                pos3 = str.size() - 1;

            if (converter(str.substr(pos0, pos1 - pos0)) &&
converter(str.substr(pos2 + 1, pos3 - pos2)))
                str.replace(pos0, pos3 - pos0 + 1, "TRUE");
            else
                str.replace(pos0, pos3 - pos0 + 1, "FALSE");
            str = analysis(str);
        } else if (str.find(" OR ") != std::string::npos) {
            pos1 = str.find(" OR ");
            pos2 = pos1 + 3;
            if (str.rfind(' ', pos1 - 1) != std::string::npos)
                pos0 = str.rfind(' ', pos1 - 1) + 1;
            else
                pos0 = 0;
            if (str.find(' ', pos2 + 1) != std::string::npos)
                pos3 = str.find(' ', pos2 + 1) - 1;
            else
                pos3 = str.size() - 1;

            if (converter(str.substr(pos0, pos1 - pos0)) ||
converter(str.substr(pos2 + 1, pos3 - pos2)))
                str.replace(pos0, pos3 - pos0 + 1, "TRUE");
            else
                str.replace(pos0, pos3 - pos0 + 1, "FALSE");
            str = analysis(str);
        }

        return str;
    }

    int main (int argc, char *argv[]) {
        std::string str;
        if (argc < 2)
            getline(std::cin, str);
        else
            str.assign(argv[1]);
        std::regex target("( )+");
        str = std::regex_replace(str, target, " ");
        target = "(\\( )+";
        str = std::regex_replace(str, target, "(");
        target = "( \\))+";
        str = std::regex_replace(str, target, ")");
        if (str[0] == ' ')
            str.erase(0, 1);
        if (str[str.size() - 1] == ' ')
            str.erase(str.size() - 1, 1);
        std::transform(str.begin(), str.end(), str.begin(), ::toupper);

        std::cout << str << " = " << analysis(str) << '\n';
        return 0;
    }
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	true and false	TRUE AND FALSE = FALSE	
2.	false or true	FALSE OR TRUE = TRUE	
3.	true and (false or true)	TRUE AND (FALSE OR TRUE) = TRUE	
4.	true and (true and (true and (true and false)))	TRUE AND (TRUE AND (TRUE AND (TRUE AND FALSE))) = FALSE	
5.	true and false or (false or false)	TRUE AND FALSE OR (FALSE OR FALSE) = FALSE	
6.	TRUE AND (FALSE OR TRUE) OR (TRUE AND FALSE)	TRUE AND (FALSE OR TRUE) OR (TRUE AND FALSE) = TRUE	
7.	TrUe Or FaLsE aNd TrUe	TRUE OR FALSE AND TRUE = TRUE	