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 _____ Аксёнова Е.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить рекурсивый подход к решению задач. Использовать рекурсивные процедуры и функции для решения поставленной задачи на языке C++.

Задание.

Построить синтаксический анализатор для понятия простое_логическое.
простое_логическое::= TRUE | FALSE | простой_идентификатор | NOT простое_логическое | (простое_логическое знак_операции простое_логическое)
простой_идентификатор::= буква
знак_операции::= AND | OR

Выполнение работы.

void isOperationCorrect:

На вход программе подается ссылки на переменные i и result, и элемент вектор vec. Сравниваем элемент вектора с номером i со строками “AND” и “OR”. Если они не равны, то добавляем 1 к переменной result.

auto isSimpleLOgicCorrect:

В лямбда-захвате передаем ссылку на вектор и ссылку на переменную result. А в параметрах указываем ссылку на i и саму функцию для рекурсивного вызова. В функции мы сравниваем элементы вектора со строками “AND”, “OR”, “NOT” или заглавной буквой. Сначала проверяем равен ли самый первый символ нулевого элемента вектора “(“. Если равен, то обрезаем первый и последний символы вектора и снова вызываем снова нашу функцию, далее вызываем функцию void isOperationCorrec для проверки операции, а затем снова нашу функцию (каждый раз проверяя на конец строки), если нет, то просто проверяем все ранее указанные строки в данном вызове. Если “AND”, “OR”, то просто увеличиваем значение i на 1. Если “NOT”, то увеличиваем i на один

и вызываем нашу функцию. Если не прошло сравнение ни с одной строкой, то увеличиваем значение result на один.

int main:

В программе объявляются переменные: строки str(в которую мы считываем исходную строку) и value (которую мы используем для записи исходной строки в вектор), вектор vec и целочисленные переменные result(используем для определения, была ли совершена ошибка), i(счетчик, которые помогает нам проходить по вектору). Считываем строку в переменную str и связываем эту строку с потоком ввода. Затем записываем строку в вектор, при этом избегая пробелы. А далее вызываем нашу лямбда-функцию. Если в переменная result не равна нулю, то выводим сообщение “The string is not correct”, а если равна, то сообщение “The string is correct”.

Разработанный программный код см. в приложении А.

Тестирование.

Для запуска тестирования в консоли прописываем команду make. Тестирование проводится по средством скрипта на языке python. Он подает программе на вход определенные строки, к которым у неё уже есть ответ. Создает файл, в который записывает результат работы пограммы. Сравнивает строку-результат с данной строкой. Выводит в консоль исход проверки, добавляя wrong, если результат неверный, и correct, если результаты совпали. Удаляет созданный файл.

Результаты тестирования см. в приложении Б.

Выводы.

Научились использовать рекурсивные процедуры и функции для решения поставленной задачи на языке C++. Была разработана программа синтаксический анализатор рекурсивно определяющая валидность строки - простого логического. Рекурсия оправдана, так как количество вызовов C(n), где n - длина строки.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab.cpp

```
#include <string>
#include <iostream>
#include <vector>
#include <sstream>



void isOperationCorrect(int& i, int& result, const std::vector<std::string>& vec) {
    if (vec[i] == "AND") {
        ++i;
        result += 0;
    }
    else if (vec[i] == "OR") {
        ++i;
        result += 0;
    }
    else {
        result += 1;
    }
}
int main(int argc, char* argv[]) {
    std::string str;
    std::vector<std::string> vec;
    int result = 0;
    std::string value;
    int i = 0;
    auto isSimpleLOgicCorrect = [&vec, &result](int& i, auto&& isSimpleLOgicCorrect)->void {
        if (vec[i][0] == '(') {
            vec[i] = vec[i].substr(1);
            vec[vec.size() - 1] = vec[vec.size() - 1].erase(vec[vec.size() - 1].size() - 1);
            if (i <= vec.size() - 1) {
                isSimpleLOgicCorrect(i, isSimpleLOgicCorrect);
                if (i <= vec.size() - 1) {
```

```
            isOperationCorrect(i, result, vec);
            if (i <= vec.size() - 1) {
                isSimpleLOgicCorrect(i, isSimpleLOgicCorrect);
            }
            else {
                result += 1;
            }
        }
        else {
            result += 1;
        }
    }
    else {
        result += 1;
    }

}
else if (vec[i] == "TRUE") {
    ++i;
    result += 0;
}
else if (vec[i] == "FALSE") {
    ++i;
    result += 0;
}
else if (vec[i].size() == 1 && isupper(vec[i][0])) {
    ++i;
    result += 0;
}
else if (vec[i] == "NOT") {
    if (i + 1 <= vec.size() - 1) {
        ++i;
        isSimpleLOgicCorrect(i, isSimpleLOgicCorrect);
    }
    else {
        result += 1;
    }
    result += 0;
}
```

```
    else {
        result += 1;
    }
};
if (argc < 2) {
    getline(std::cin, str, '\n');
}
else {
    for (int j = 1; argv[j]; j++) {
        for (int k = 0; argv[j][k]; k++) {
            str.push_back(argv[j][k]);
        }
        if (argv[j + 1]) {
            str.push_back(' ');
        }
    }
}

std::stringstream ss(str);

while (ss >> value)
{
    vec.push_back(value);

    if (ss.peek() == ' ') {
        ss.ignore();
    }
}



isSimpleLOgicCorrect(i, isSimpleLOgicCorrect);

if (i != vec.size()) {
    result += 1;
}

if (!result) {
```

```
        std::cout << "The string is correct" << '\n';
    }
    else {
        std::cout << "The string is not correct" << '\n';
    }

}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Результат проверки
1.	TRUE	The string is correct	correct
2.	A	The string is correct	correct
3.	NOT NOT NOT NOT A	The string is correct	correct
4.	(TRUE AND FALSE)	The string is correct	correct
5.	TRUE AND NOT A	The string is not correct	correct
6.	(NOT A)	The string is not correct	correct
7.	NOT FALSE	The string is correct	correct
8.	(NOT AND NOT NOT A)	The string is not correct	correct
9.	False	The string is not correct	correct
10.	NOT (NOT NOT FALSE OR NOT NOT NOT A)	The string is correct	correct
11.	(NOT A AND FALSE	The string is not correct	correct
12.	(FALSE AND TRUE)	The string is not correct	correct
13.	NOT A AND FALSE)	The string is not correct	correct

14.	(AND)	The string is not correct	correct