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. 9381 _____ Камакин Д.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++. Разработать программу, использующую рекурсию.

Задание.

Вариант 6.

Построить синтаксический анализатор для понятия *простое выражение*.
Простое_выражение:: = простой_идентификатор | (простое_выражение знак_операции простое_выражение)
Простой_идентификатор:: = буква
Знак_операции:: = - | + | *

Описание алгоритма.

В функции main() пользователю предлагается выбрать способ считывания данных (поддерживается ввод с клавиатуры и файла). Затем вызывается рекурсивная функция stringAnalyze(), аргументами которой являются считанная строка, количество открытых скобок, знаков операции и букв до текущего символа.

bool stringAnalyze(string input, int bracketsOpen, int alphaBefore, int symbolBefore) — сигнатура рекурсивной функции (синтаксический анализатор). string input — исходная строка, int bracketsOpen — количество открытых скобок, int alphaBefore — количество букв до текущего момента, int symbolBefore — количество знаков операции до текущего момента.

Далее синтаксический анализатор посимвольно рекурсивно проверяет строку на соответствие условию, выводя промежуточную информацию.

Если функция наткнулась на букву, то она проверяет, были ли скобки или другие буквы до неё. Если не было скобок и она единственная в строке, то возвращаем истину, иначе — ложь. Увеличиваем счётчик alphaBefore на 1, если буква в строке не одна, до неё не было других букв и скобка открыта.

Если встретили символ открывающейся скобки, увеличиваем счётчик bracketsOpen на 1.

Если встретили символ закрывающейся скобки, то проверяем, были ли до неё буквы/знаки операции. Уменьшаем счётчик открытых скобок и количества знаков операции.

Если встретили символ знака операции, проверяем, был ли он в скобке, также увеличиваем счётчик symbolBefore и уменьшаем alphaBefore.

В конце функции отрезаем текущий символ от строки и производим рекурсивный вызов.

Вывод.

В ходе выполнения лабораторной работы был изучен такой вид программ, как синтаксические анализаторы. Была реализована программа, которая анализирует строку рекурсивным методом, определяя соответствие определению.

Входные данные	Результат
((a+b)-(c+b))	Would you like to read from the file? (1/0)? 0 Input the string: ((a+b)-(c+b)) Found '(' and bracketsOpen = 1 *Found '(' and bracketsOpen = 2 **Found an alpha 'a' alphaBefore = 1 ***Found a symbol '+' bracketsOpen = 2 and symbolBefore = 1 ****Found an alpha 'b' alphaBefore = 1 *****Found ')' , bracketsOpen = 1 and alphaBefore = 1 ******Found a symbol '-' bracketsOpen = 1 and symbolBefore = 1 *******Found '(' and bracketsOpen = 2 ********Found an alpha 'c' alphaBefore = 1 *********Found a symbol '+' bracketsOpen = 2 and symbolBefore = 2 **********Found an alpha 'b' alphaBefore = 1 ***********Found ')' , bracketsOpen = 1 and alphaBefore = 1 ************Found ')' , bracketsOpen = 0 and alphaBefore = 1 *************Done. The string is correct Correct
A	Correct
(a+b))	Not correct
(a+(b-c)*c)	Not correct
((((b+h))))	Not correct

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>

#include <string>

#include <fstream>


using namespace std;


static int recursionLevel = 0;


/*

 * Построить синтаксический анализатор для понятия простое
выражение.

 *    Простое_выражение:: = простой_идентификатор |
(простое_выражение знак_операции простое_выражение)

 *    Простой_идентификатор:: = буква (alpha)

 *    Знак_операции:: = - | + | * (symbol)

 */


void printRecursion(int level) { // Outputs '*' for recursion
depth

    for (int i = 0; i < level; i++)

        cout << '*';

}


bool stringAnalyze(string input, int bracketsOpen, int
alphaBefore, int symbolBefore) {

    if (input.empty() && bracketsOpen != 0)  { // The string
is over but bracket(s) wasn't closed

        printRecursion(recursionLevel);

        cout << "Invalid brackets" << '\n';

        recursionLevel--;

        return false;

    }


    if (input.empty()) { // The string was parsed and brackets
closed
```

```
        printRecursion(recursionLevel);
        cout << "Done. The string is correct" << '\n';
        recursionLevel--;
        return true;
    }


    if (isalpha(input[0])) { // Case for an alpha
        if ((!bracketsOpen && input.size() > 1) || alphaBefore
> 1) {  // No brackets but more than 1 alpha
            printRecursion(recursionLevel);
            cout << "You have more than 1 simple id (an alpha)
in a row" << '\n';
            recursionLevel--;
            return false;
        } else if (input.size() == 1 && !bracketsOpen) { //
Only one symbol
            printRecursion(recursionLevel);
            cout << "Found only one alpha so it's correct." <<
'\n';
            recursionLevel--;
            return true;
        }


        alphaBefore++;
        printRecursion(recursionLevel);
        cout << "Found an alpha '" << input[0]  << "'
alphaBefore = " << alphaBefore << '\n';
    } else if (input[0] == '(') {
        bracketsOpen++;
        printRecursion(recursionLevel);
        cout << "Found '(' and bracketsOpen = " <<
bracketsOpen << '\n';
    } else if (input[0] == ')') {
        if (alphaBefore != 1 || symbolBefore < 1) { // (alpha
symbol alpha) -> check for alpha and symbol
            printRecursion(recursionLevel);
            cout << "Found ')' but there were no alpha. Wrong
brackets, not correct." << '\n';
            recursionLevel--;
```

```
            return false;
        }


        bracketsOpen--;
        symbolBefore--;
        printRecursion(recursionLevel);
        cout << "Found ')' , bracketsOpen = " << bracketsOpen
<< " and alphaBefore = " << alphaBefore << '\n';
    } else if (input[0] == '+' || input[0] == '-' || input[0]
== '*')  { // Case for the operations
        symbolBefore++;


        if (bracketsOpen < 1 || alphaBefore < 1 ||
(symbolBefore >= 2 && bracketsOpen < 2)) { // An operation not
in brackets
            printRecursion(recursionLevel);
            cout << "Found a symbol but it's not in brackets,
symbolBefore = "  << symbolBefore << '\n';
            recursionLevel--;
            return false;
        }


        alphaBefore--;
        printRecursion(recursionLevel);
        cout << "Found a symbol '" << input[0] << "'
bracketsOpen = " << bracketsOpen <<
                    " and symbolBefore = " << symbolBefore <<
'\n';
    } else {
        printRecursion(recursionLevel);
        cout << "Invalid symbol starting here: " << input <<
'\n';
        recursionLevel--;
        return false;
    }


    input = input.substr(1, input.size() - 1);
    recursionLevel++;
    return stringAnalyze(input, bracketsOpen, alphaBefore,
symbolBefore);
```

```
}

int main() {
    string input, fileName;

    bool isFile;

    cout << "Would you like to read from the file? (1/0)?" <<
'\n';

    cin >> isFile;
    cin.get();

    if (isFile) {
        cout << "Input the full path to the isFile: " << '\n';
        getline(cin, fileName);

        ifstream myFile(fileName);
        if (myFile.fail()) {
            cout << "Couldn't open the file. Try again" << '\
n';
            return 0;
        }

        getline(myFile, input);
        cout << "The line from the file: " << input << '\n';
    } else {
        cout << "Input the string: " << '\n';
        getline(cin, input);
    }

    if (!input.empty())
        cout << (stringAnalyze(input, 0, 0, 0) ? "Correct" :
"Not correct") << '\n';
    else
        cout << "Something went wrong" << '\n';
    return 0;
```

```
}
```