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

ОТЧЁТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия

Студент гр. 8381 _____ Почаев Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы

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

Задание

Вариант 21

Построить синтаксический анализатор для понятия *скобки*.

$$\text{скобки} ::= \begin{cases} \text{квадратные} \\ \text{круглые} \end{cases} \qquad \text{квадратные} ::= \begin{cases} [[\text{квадратные}](\text{круглые})] \\ B \end{cases}$$

$$\text{круглые} ::= \begin{cases} ((\text{круглые})[\text{квадратные}]) \\ A \end{cases}$$

Основные теоретические положения

Взаимная рекурсия

Взаимная (косвенная) рекурсия — это вид рекурсии, когда два математических или программных объекта, таких как функции или типы данных, определяются в терминах друг друга

Строки

При выполнении данной лабораторной работы были использованы взаимно-рекурсивные функции: последовательность символов, называемой *скобки*, в которой присутствуют две взаимно-рекурсивные части: квадратные определяются через круглые, и наоборот, круглые − через квадратные. В простейшем случае квадратные есть символ «B», а круглые - «A». Написанная программа - синтаксический анализатор определяет, является ли заданная (входная) последовательность символов скобками или нет. В случае ответа «нет» сообщается место и причина ошибки.

Выполнение работы.

Разработка программы велась на базе операционной системы Ubuntu 18.04

с использованием специализированной IDE QtCreator. Программа написана на языке C++ с использованием фреймворка Qt. Для графического оформления программы использовалась графическая оболочка ColinDuquesnoy/QdarkStyleSheet по открытой лицензии с GitHub, стилистические дополнения на CSS и иконки из свободных источников.

В начале был разработан графический интерфейс приложения: заголовок с названием программы и коротким пояснением для пользователя, ниже находится поле ввода тестируемой строки и кнопка, отправляющая её на тестирование. Ниже расположена кнопка, открывающая системное окно выбора текстового файла формата `*.txt` из которого также возможно произвести чтение строки. При удачном и неудачном результата тестирования программа создаёт информационное окно: в случае первого - об удачном тестировании, во втором - сообщение об ошибке, её место в строке и позицию. Также предусмотрены ситуации, когда пользователь не ввёл строку и отправил на проверку, или же не выбрал файл в системном диалоговом окне: в таком случае выдаются соответственные информационные сообщения.

После оправки строки на проверку производится остальная подготовительная работа: рядом с исходным кодом программы, на директорию выше, создаётся файл с логами, в который впоследствии записываются имена функций в порядке из вызова и также конечный результат. Также открывается поток на запись и он вместе и полученной от пользователя строкой передаются в функцию проверки строки `checkString(inputString, logStream)`.

В теле функции `checkString(inputString, logStream)` происходит конвертация строки из формата `QStrinf` в `std::string`, обнуление счётчика текущей позиции проверки (для случая возникновения ошибки) и вызов булевой функции - основной части программы `Brackets(inputStr, pos, log))`. Данная функция производит последовательный вызов двух других (парных) булевых функций: Round+ и Square+, определяющих, являются ли текущие последовательности частью *круглые* или *квадратные* соответственно. Каждая из функ-

ций `Round` и `Square` в свою очередь вызывают парную к себе (`Square` и `Round` соответственно).

Также в данной программе используются вспомогательные функции `step`, которая „отрезает“ первый символ строки, что выполняет „шаг вперёд“ , и `Mist`, которая используется для вывода ошибки и сообщении, где именно она произошла. Для последнего данная функция принимает на вход название вызвавшей функции.

Ход работы обеих функций `Round` и `Square` аналогичен и отличается лишь в последовательности проверяемых скобок. В местах стационарных скобок `[]` и `()` происходит проверка на их соответствие заданному шаблону и „два шага вперёд“ , а для проверки вложенных *квадратных* и *круглых* скобок используются соответствующие рекурсивные вызовы функций.

Тестирование

Результаты выполнения программы на некоторых тестовых данных приведены в таблице 1.

№	Input	Result
1	[[B](A)]	The result is true: [[B](A)] - correct
2	[[[[B](A)]](((A)[B]))]	The result is true: [[[[B](A)]](((A)[B]))] - correct
3	[[[B](A)]]	String is incorrect! Mistake is in: Round parentheses with: ”[B](A)]]”on 2 position
4	[[[[B](A)]]((A)[B]))]	String is incorrect! Mistake is in: Round parentheses with: ”(A)[B]))]”on 12 position

1: Testing results

Выводы

В процессе выполнения лабораторной работы были изучены основы работы с рекурсией и написание программ с использованием взаимно-рекурсивных функций и процедур. Были получены навыки отладки программы, использующей рекурсию, и изучены различные пути взаимодействия с ней, позволяющие выстроить логический алгоритм работы. Также в ходе данной работы были по-

лучены знания по составлению синтаксического анализатора текста.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "mainwindow.h"
#include "visualfunc.h"
#include "universalqtheader.h"
#include "basicheaders.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    connectVisualLib();
    w.show();

    return a.exec();
}
```

Название файла: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "universalqtheader.h"
#include "stringchecker.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_fileChoose_clicked()
{
    QString fileName = QFileDialog::getOpenFileName(this,
    tr("Open TXT File"),  QDir::homePath(),
    tr("TXT text (*.txt);;All Files (*)"));
    if (fileName == nullptr)
    {
        QMessageBox::warning(this,
        "Warning!",
        "Please choose text file for work.");
        return;
    }

    QString logFilePath = "/";
```

```
    QFile file(fileName);
    QString fileString;

    // making log file in path of pr
    QFile log("log.txt");
    log.open(QIODevice::ReadWrite);
    QTextStream logStream(&log);

    // only reading and interpretation as a text
    if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        fileString = stream.readLine();
    }

    file.close();

    std::string inputString = fileString.toUtf8().constData();

    checkString(inputString, logStream);

    log.close();
}

void MainWindow::on_inputString_clicked()
{
    // convert QString to srd::string, using UTF-8
    std::string inputString = \
            ui->inputLine->text().toUtf8().constData();
    if (inputString.empty()) {
        QMessageBox::warning(this,
        "Warning!",
        "Please input the string.");
        return;
    }

    // making log file in path of pr
    QString logFilePath = QDir::currentPath();
    logFilePath.chop(logFilePath.length() - \
                     logFilePath.lastIndexOf(QChar('/')));
    QFile log(logFilePath + "/log.txt");
    log.open(QIODevice::ReadWrite | \
             QIODevice::Truncate | QIODevice::Text);
    QTextStream logStream(&log);

    checkString(inputString, logStream);

    log.close();
}
```

Название файла: stringchecker.cpp

```
#include "basicheaders.h"
#include "universalqtheader.h"

// Parentheses
```

```
bool Brackets(std::string str, int & pos, QTextStream & log);
// Square brackets
bool Square(std::string str, int & pos, QTextStream & log);
// Round brackets
bool Round(std::string str, int & pos, QTextStream & log);
// Go to the next line character
void step(int & pos, QTextStream & log);
// errors
void Error(const std::string & mistake, std::string str, \
          int pos, QTextStream & log);

int checkString(std::string inputStr, QTextStream & log) {
    int pos = 0;
    std::string successStr = inputStr;
    if (!Brackets(inputStr, pos, log)) {
        successStr = "The result is true: " + successStr + \
                " - correct\n";
        std::cout << successStr;
        log << QString::fromStdString(successStr);
        QMessageBox::information(0, "Success", \
                                 QString::fromStdString(successStr));
    }
    pos = 0;

    return 0;
}

bool Brackets(std::string str, int & pos, QTextStream & log) {
    log << "Brackets func!\n";
    if ((str[pos]=='[') || (str[pos]=='B')) {
        if (Square(str, pos, log))  // is sequence in square paired
            return 1;
        else
            return 0;
    } else if ((str[pos]=='(') || (str[pos]=='A')) {
        if (Round(str, pos, log)) // is sequence in round paired
            return 1;
        else
            return 0;
    }

    Error("Brackets", str, pos, log); // Error processing

    return 1;
}

bool Square(std::string str, int & pos, QTextStream & log) {
    log << "Square func!\n";
    if (str[pos] == '[' && str[pos + 1] == '[') {
        step(pos, log); // go to the next symbol in str
        step(pos, log);
        if (Square(str, pos, log)) // check for paired round #1
            return 1;
        if (str[pos] == ']' && str[pos + 1] == '(') {
            step(pos, log);
```

```
            step(pos, log);
            if (Round(str, pos, log))
                return 1;
            if (str[pos] == ')' && str[pos + 1] == ']') {
                step(pos, log);
                step(pos, log);
                return 0;
            }
        } else {
            Error("square parantheses", str, pos, log);
            return 1;
        }
    }

    if (str[pos] == 'B') {
        step(pos, log);
        return 0;
    }

    Error("square parantheses", str, pos, log);

    return 1;
}

bool Round(std::string str, int & pos, QTextStream & log) {
    log << "Round func!\n";
    if (str[pos] == '(' && str[pos + 1] == '(') {
        step(pos, log); // go to the next symbol in str
        step(pos, log);
        if (Round(str, pos, log)) // check for paired round #1
            return 1;
        if (str[pos] == ')' && str[pos + 1] == '[') {
            step(pos, log);
            step(pos, log);
            if (Square(str, pos, log))
                return 1;
            if (str[pos] == ']' && str[pos + 1] == ')') {
                step(pos, log);
                step(pos, log);
                return 0;
            }
        } else {
            Error("round parantheses", str, pos, log);
            return 1;
        }
    }

    if (str[pos] == 'A') {
        step(pos, log);
        return 0;
    }

    Error("round parantheses", str, pos, log);

    return 1;
```

```
}

void step(int & pos, QTextStream & log) {
    log << "step func!\n";
    pos++;
}

void Error(const std::string & mistake, std::string str, \
          int pos, QTextStream & log) {
    log << "Error func!\n";
    std::string resStr = "String is incorrect! Mistake is in: \n" + \
            mistake + ". Successful part:\n" + "\"" + \
                str.substr(0, pos) + "\" \n" "Mistake on " + std::to_string(po
    std::cout << resStr;
    log << QString::fromStdString(resStr);
    QMessageBox::information(0, "Mistake", QString::fromStdString(resStr));
}
```

Название файла: visualfunc.cpp

```
#ifndef VISUALFUNC_CPP
#define VISUALFUNC_CPP
#include <QFile>
#include <QTextStream>
#include <QApplication>
#include <iostream>

void connectVisualLib()
{
    QFile f(":qdarkstyle/style.qss");
    if (!f.exists())
    {
        std::cout << "Unable to set stylesheet, file not found\n";
    }
    else
    {
        f.open(QFile::ReadOnly | QFile::Text);
        QTextStream ts(&f);
        qApp->setStyleSheet(ts.readAll());
    }
}
#endif // VISUALFUNC_CPP
```