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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсивная обработка иерархических списков
Вариант 15

Студент гр. 8304 _____ Щука А. А.

Преподаватель _____ Фиалковский М. С.

Санкт-Петербург
2019

Цель работы.

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков, с использованием базовых функций их рекурсивной обработки.

Постановка задачи.

1) проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
2) разработать программу, использующую рекурсию;
3) сопоставить рекурсивное решение с итеративным решением задачи;
4) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

Вариант 15: проверить структурную идентичность двух иерархических списков (списки структурно идентичны, если их устройство (скобочная структура и количество элементов в соответствующих подсписках одинаково, при этом атомы могут отличаться);

Описание алгоритма.

Для начала программа должна рекурсивно считать данные, проверить их корректность и занести их в список. Для этого считываем очередной символ строки. Если это атом, добавляем его к иерархическому списку, если это список – рекурсивно заносим его в список.

Далее для определения структурной идентичности двух иерархических списков необходимо, чтобы при их параллельном переборе они одновременно указывали на атом или список. Если один из элементов списка указывает на атом или на список, а другой указывает на следующий парный элемент списка, то тогда списки не идентичны.

```
сравнение_списков (структура_списка1, структура_списка2) {
        если (размер_списка1 !=  размер_списка2)
            вернуть false
```

```
        цикл по всем вложенным спискам {
    если (атом_списка1 и атом_списка2)
        следующая итерация цикла
    еще если (!атом_списка1 и !атом_списка2)
        если (сравнение_списков (вложенный_список1, вложенный_спискок2)
            следующая итерация цикла
        иначе
            вернуть false
    иначе
        вернуть false
    }
    вернуть true
}
```

Спецификация программы.

Программа предназначена для проверки идентичности двух иерархических списков.

Программа написана на языке C++ с использованием фреймворка Qt. Входными данными являются символы английского алфавита и скобки - считываются из полей lineEdit или из файла. Выходными данными являются промежуточные значения вычисления выражения и конечный результат. Данные выводятся в qDebug(), результат показывается в всплывающем окне.

Тестирование.

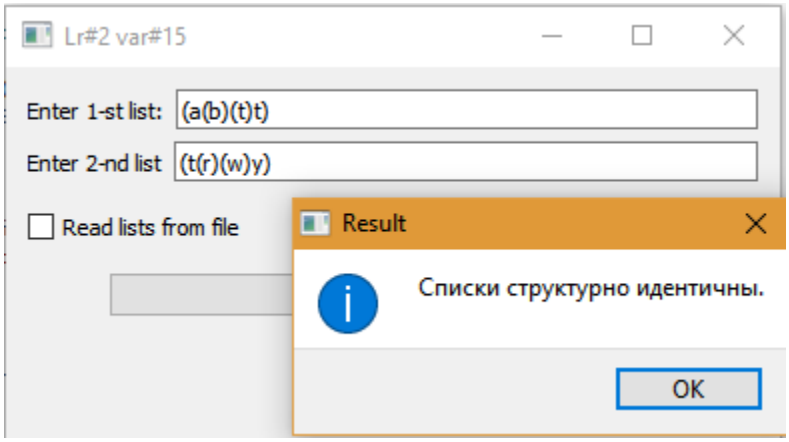

Рисунок 1- сравнение списков

Остальные тесты приведены в табл. 1.

Таблица 1 – результаты тестирования программы

Ввод	Результат
(qwe()) (abc())	Списки идентичны
() ()	Списки идентичны
(qwer) (qwer)	Списки идентичны
((qwer()(vddkbk))(ervnvn()ekvnnlkewnvkl(ifuhiuvhi)evenn)wevnjvndn(()()edv)) ((qwer()(vddkbk))(ervnvn()ekvnnlkewnvkl(ifuhiuvhi)evenn)wevnjvndn(()()edv))	Списки идентичны
(weq(weq((weq)(weq)(we))egdb)egdb) (weq(weq((weq)(weq)(we))egdb)egdb)	Списки идентичны
(qwe) (qwer)	Разное количество элементов в скобках
(qwe()ty) (qwerty)	Разные элементы в списках
qwe()rty qwe()rty	Строка должна начинаться с открывающей скобки
(qwe(rty))uio (qwe(rty))uio	Строка должна заканчиваться закрывающей скобкой
(qwe(qwerq)(qwer) (qwe(qwerq)(qwer)	Несоответствие количества открывающих и закрывающих скобок

Анализ алгоритма.

Для экономии памяти передается указатели на списоки, что позволяет избежать копирования. Алгоритм работает за линейное время. Недостаток рекурсивного алгоритма – ограниченный стек вызовов функций, что в свою очередь накладывает ограничение на количество вложенных списков, а также затраты производительности на вызов функций.

Описание функций и СД.

Класс-реализация иерархического списка MyList содержит вектор указателей на вложенные списки, флаг, для определения атома, значение атома.

Метод класса для проверки входных данных на корректность:

```
bool MyList::checkStr(const std::string& str) const
```

Принимает на вход ссылку на строку, проверяет размер и структуру строки, возвращает true, если строка корректна, и false в ином случае.

Метод класса для создания иерархического списка:

```
bool MyList::buildList(const std::string& str)
```

Принимает на вход ссылку на строку, проверяет корректность строки и рекурсивно создает список. Если элемент - атом, тогда он добавляется в вектор указателей, если элемент - список, тогда он создается рекурсивно и добавляется в вектор указателей.

Статический метод класса для сравнения двух списков:

```
static bool compareList(MyList* firstList, MyList* secondList, size_t depth = 0);
```

Принимает на вход два списка, глубину рекурсии для отладки и возвращает true, если списки идентичны, и false в ином случае. Сначала сравниваются размеры списков, затем функция итеративно проходит по вложенным спискам. Если оба списка - атомы, тогда всё нормально. Если оба списка - вложенные спис-

ки - функция вызывается рекурсивно для вложенных списков. Если элементы различны - списки не идентичны.

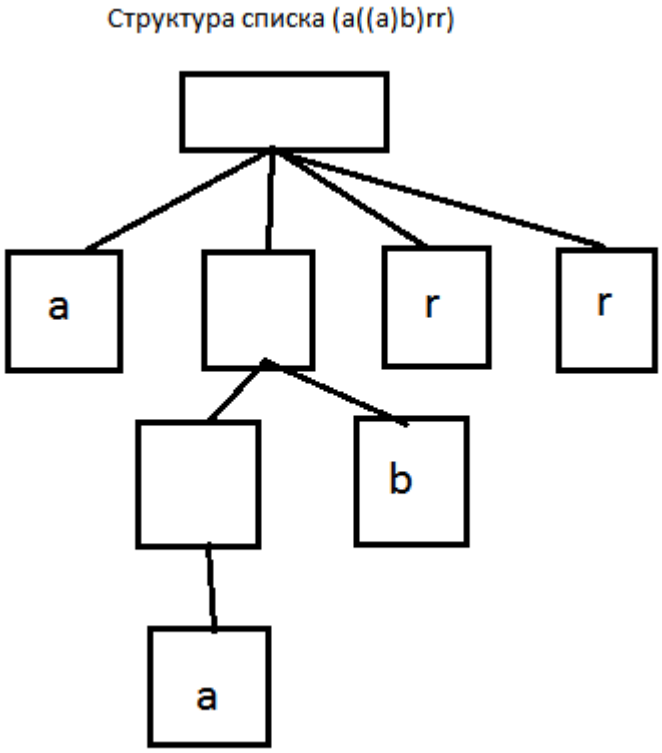

Рисунок 2 – структура иерархического списка

Выводы.

В ходе работы я вспомнил работу с классами и контейнерами в C++. Научился решать задачи обработки иерархических списков, с использованием базовых функций их рекурсивной обработки и сравнивать два иерархических списка на структурную идентичность.

Приложение А.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextStream>
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QDebug>
#include <QString>
#include <QDir>
#include <QStringList>
#include <QFileSystemModel>
#include <string>
#include "mylist.h"


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_readFile_checkBox_clicked();

    void on_compare_pushButton_clicked();

    void on_test_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    QTextStream* in;
    QFile* file;
    QDir* dir;
};

#endif // MAINWINDOW_H
```

mylist.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextStream>
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QDebug>
#include <QString>
#include <QDir>
#include <QStringList>
#include <QFileSystemModel>
#include <string>
#include "mylist.h"


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_readFile_checkBox_clicked();

    void on_compare_pushButton_clicked();

    void on_test_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    QTextStream* in;
    QFile* file;
    QDir* dir;
};

#endif // MAINWINDOW_H
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>


//var #15


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

    MainWindow window;
    window.setWindowTitle("Lr#2 var#15");
    window.show();

    return app.exec();
}
```

mainwindow.cpp

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    file = new QFile;
    in = new QTextStream;
    dir = new QDir;
}


MainWindow::~MainWindow()
{
    delete ui;
    delete in;
    delete file;
    delete dir;
}


void MainWindow::on_readFile_checkBox_clicked()
{
    //выбор считывания из файла или из окна
    if (ui->readFile_checkBox->isChecked()) {
        file->close();

        ui->textLabel_0->setEnabled(false);
        ui->inputFirstList_lineEdit->setEnabled(false);
        ui->textLabel_1->setEnabled(false);
        ui->inputSecondList_lineEdit->setEnabled(false);

        QString fileName = QFileDialog::getOpenFileName(this,
                                                         "Open file");
```

```
        file->close();
        file->setFileName(fileName);
        file->open(QFile::ReadOnly | QFile::Text);
        in->setDevice(file);

        ui->openFile_textLabel->setText(fileName);
    }
    else {
        file->close();

        ui->textLabel_0->setEnabled(true);
        ui->inputFirstList_lineEdit->setEnabled(true);
        ui->textLabel_1->setEnabled(true);
        ui->inputSecondList_lineEdit->setEnabled(true);
        ui->openFile_textLabel->setText("none");
    }
}


void MainWindow::on_compare_pushButton_clicked()
{
    qDebug();
    qDebug() << "Сравнение списков:";

    std::string firstStr = "";
    std::string secondStr = "";

    //считывание строк из файла или из окна
    if (ui->readFile_checkBox->isChecked()) {
        firstStr = in->readLine().toStdString();
        secondStr = in->readLine().toStdString();
    }
    else {
        firstStr = ui->inputFirstList_lineEdit->text().toStdString();
        secondStr = ui->inputSecondList_lineEdit->text().toStdString();
    }

    //создание списков, проверка на корректность
    MyList firstList;
    MyList secondList;

    if (firstList.buildList(firstStr) &&
            secondList.buildList(secondStr)) {
        qDebug() << "Строки корректны, списки созданы";
    }
    else {
        qDebug() << "Строки некорректны!";
        QMessageBox::critical(this, "Result", "Входные данные неккоректны.");
        return;
    }

    qDebug() << "__Считанные списки__:";
    qDebug() << "Первый список:" << firstList;
    qDebug() << "Второй список:" << secondList;

    //сравнение списков
    if (MyList::compareList(&firstList, &secondList)) {
        qDebug() << "Списки структурно идентичны.";
        QMessageBox::information(this, "Result", "Списки структурно идентичны.");
    }
    else {
        qDebug() << "Списки НЕ структурно идентичны.";
        QMessageBox::warning(this, "Result", "Списки НЕ структурно идентичны.");
    }

    ui->readFile_checkBox->setChecked(false);
    ui->textLabel_0->setEnabled(true);
    ui->textLabel_1->setEnabled(true);
    ui->inputFirstList_lineEdit->setEnabled(true);
    ui->inputSecondList_lineEdit->setEnabled(true);
    ui->openFile_textLabel->setText("none");
}


void MainWindow::on_test_pushButton_clicked()
{
    /*
     * Функция тестирования. Тестовые данные считываются из папки Tests
    */
```

```
    qDebug() << "Тестирование.";

    std::string firstStr = "";
    std::string secondStr = "";
    dir->cd(QApplication::applicationDirPath() + "/Tests");
    QStringList listFiles = dir->entryList(QStringList("*.txt"), QDir::Files);

    for (auto fileName : listFiles) {
        if (fileName == "." || fileName == "..")
            continue;

        qDebug();
        qDebug() << "Тестовые данные из файла:" << fileName;

        file->close();
        file->setFileName(dir->path() + "/" + fileName);
        file->open(QFile::ReadOnly | QFile::Text);
        in->setDevice(file);

        while (!in->atEnd()) {
            firstStr = in->readLine().toStdString();
            secondStr = in->readLine().toStdString();

            MyList firstList;
            MyList secondList;

            if (firstList.buildList(firstStr) &&
                    secondList.buildList(secondStr)) {
                qDebug() << "Строки корректны, списки созданы";
            }
            else {
                qDebug() << "Строки некорректны!";
                continue;
            }

            qDebug() << "__Считанные списки__:";
            qDebug() << "Первый список:" << firstList;
            qDebug() << "Второй список:" << secondList;

            //сравнение списков
            if (MyList::compareList(&firstList, &secondList)) {
                qDebug() << "_Списки структурно идентичны._";
            }
            else {
                qDebug() << "_Списки НЕ структурно идентичны._";
            }
            qDebug();
        }
    }
}
```

mylist.cpp

```
#include "mylist.h"

MyList::MyList(QObject *parent) : QObject(parent)
{
    /*
     * По умолчанию объект класса является списком
    */

    isAtom = false;
    atom = 0;
}


MyList::~MyList()
{
    /*
     * Освобождение выделенной под список памяти
    */

    for (auto elem : childArray)
        delete elem;
}


QDebug& operator<<(QDebug& out, const MyList& list)
```

```
{
    /*
     * Перегрузка оператора вывода для отладки пограммы
    */

    out << "(";

    for (auto elem:list.childArray) {
        if (elem->isAtom)
            out << elem->atom;
        else
            out << *elem;
    }

    out << ")";
    return out;
}


bool MyList::checkStr(const std::string& str) const
{
    /*
     * Функция проверки корректности входных данных,
     * принимает на вход ссылку на строку, проверяет размер и структуру строки,
     * возвращает true, если строка корректна, и false в ином случае
    */

    qDebug() << "Проверка на корректность:" << str.c_str();
    int countBracket = 0;

    if (str.size() < 2)
        return false;

    size_t i;
    for (i = 0; i < str.size(); ++i) {
        char elem = str[i];
        if (elem == '(')
            ++countBracket;
        else if (elem == ')')
            --countBracket;
        else if (!isalpha(elem))
            return false;
    }

    if (countBracket > 0 || i != str.size()) {
        return false;
    }

    qDebug() << "Строка корректна.";
    return true;
}


void MyList::createAtom(const char ch)
{
    /*
     * Объект класса становится листом списка - атомом
    */

    this->atom = ch;
    this->isAtom = true;
}


bool MyList::buildList(const std::string& str)
{
    /*
     * Функция создания иерархического списка. Принимает на вход ссылку
     * на строку, проверяет корректность строки и рекурсивно создает список. Если
     * элемент - атом, тогда он добавляется в вектор указателей, если элемент - список,
     * тогда он создается рекурсивно и добавляется в вектор указателей.
    */

    qDebug() << "В список добавляется содержимое следующих скобок:" << str.c_str();

    if (!checkStr(str))
        return false;

    size_t i = 1;
```

```
    int countBracket = 0;

    while (i < str.size() - 1) {
        if (isalpha(str[i])) {
            qDebug() << "Добавялется атом:" << str[i];

            MyList *tmp = new MyList;
            tmp->createAtom(str[i]);
            childArray.push_back(tmp);
        }
        else {
            size_t j = i;
            ++countBracket;
            ++i;
            while (countBracket != 0) {
                if (str[i] == '(')
                    ++countBracket;
                else if (str[i] == ')')
                    --countBracket;
                ++i;
            }
            MyList* tmp = new MyList;
            tmp->buildList(str.substr(j, i-j));
            childArray.push_back(tmp);
            --i;
        }
        ++i;
    }

    return true;
}


size_t MyList::size() const
{
    /*
     * Функция возвращает количество вложенных списков и атомов
    */

    return childArray.size();
}


char MyList::getAtom() const
{
    /*
     * Функция возвращает значение атома
    */

    return atom;
}


bool MyList::compareList(MyList *firstList, MyList *secondList, size_t depth)
{
    /*
     * Функция сравнения двух списков.
     * Принимает на вход два списка, глубину рекурсии для отладки и
     * возвращает true, если списки идентичны, и false в ином случае.
     *
     * Сначала сравниваются размеры списков, затем функция итеративно проходит по
     * вложенным спискам. Если оба списка - атомы, тогда всё нормально. Если оба списка -
     * вложенные списки - функция вызывается рекурсивно для вложенных списков. Если элементы различ-
ны -
     * списки неидентичны.
    */

    std::string dbgStr = "";
    for(size_t i = 0; i < depth; ++i)
        dbgStr += " ";

    qDebug() << dbgStr.c_str() << "Сравнение списков";

    if (firstList->size() != secondList->size()) {
        qDebug() << dbgStr.c_str() << "Списки различной длины";
        return false;
    }

    for (size_t i = 0; i < firstList->size(); ++i) {
```

```
        if (firstList->childArray[i]->isAtom && secondList->childArray[i]->isAtom) {
            qDebug() << dbgStr.c_str() << "Сравниваются элементы:" <<
                        firstList->childArray[i]->getAtom() << "и" << secondList->childArray[i]-
>getAtom();
        }
        else if (!firstList->childArray[i]->isAtom && !secondList->childArray[i]->isAtom) {
            qDebug() << dbgStr.c_str() << "Заход во внутренний список";

            if (compareList(firstList->childArray[i], secondList->childArray[i],
                            depth + 1)) {
                qDebug() << dbgStr.c_str() << "Внутренние списки идентичны";
            }
            else {
                qDebug() << dbgStr.c_str() << "Внутренние списки различны";
                return false;
            }
        }
        else {
            qDebug() << dbgStr.c_str() << "Различные элементы в списках";
            return false;
        }
    }
    qDebug() << dbgStr.c_str() << "Списки идентичны.";
    return true;
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>387</width>
    <height>185</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Ignored" vsizetype="Ignored">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
       <widget class="QLabel" name="textLabel_0">
        <property name="text">
         <string>Enter 1-st list:</string>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QLineEdit" name="inputFirstList_lineEdit"/>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout_3">
      <item>
       <widget class="QLabel" name="textLabel_1">
        <property name="text">
         <string>Enter 2-nd list</string>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QLineEdit" name="inputSecondList_lineEdit"/>
      </item>
     </layout>
    </item>
```

```
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_5">
  <item>
   <widget class="QCheckBox" name="readFile_checkBox">
    <property name="text">
     <string>Read lists from file</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_3">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QLabel" name="label">
    <property name="text">
     <string>File path:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLabel" name="openFile_textLabel">
    <property name="text">
     <string>none</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_2">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Preferred</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="compare_pushButton">
    <property name="styleSheet">
     <string notr="true"/>
    </property>
    <property name="text">
     <string>Compare</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_2">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Preferred</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
```

```
          <width>40</width>
          <height>20</height>
         </size>
        </property>
       </spacer>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout_4">
      <item>
       <spacer name="horizontalSpacer_4">
        <property name="orientation">
         <enum>Qt::Horizontal</enum>
        </property>
        <property name="sizeHint" stdset="0">
         <size>
          <width>40</width>
          <height>20</height>
         </size>
        </property>
       </spacer>
      </item>
      <item>
       <widget class="QPushButton" name="test_pushButton">
        <property name="text">
         <string>Test</string>
        </property>
       </widget>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menuBar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>387</width>
     <height>21</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```