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

ОТЧЕТ

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

Студентка гр. 8381 _____ Лисок М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивной обработки списков, изучить особенности реализации иерархического списка на языке программирования C++. Разработать программу, использующую иерархические списки и их рекурсивную обработку, анализирующую корректность выражения.

Задание.

Вариант 12:

проверить идентичность двух иерархических списков.

Основные теоретические положения.

Согласно рекурсивному определению, иерархический список – такой список, элементами которого могут быть иерархические списки. Для обработки иерархического списка удобно использовать рекурсивные функции, так как он представляет собой множество линейный списков, между которыми установлены связи, иерархия.

В качестве примера наглядно демонстрирующие иерархические списки, на рисунке 1 представлен список, соответствующий сокращенной *((a b)c d)*.

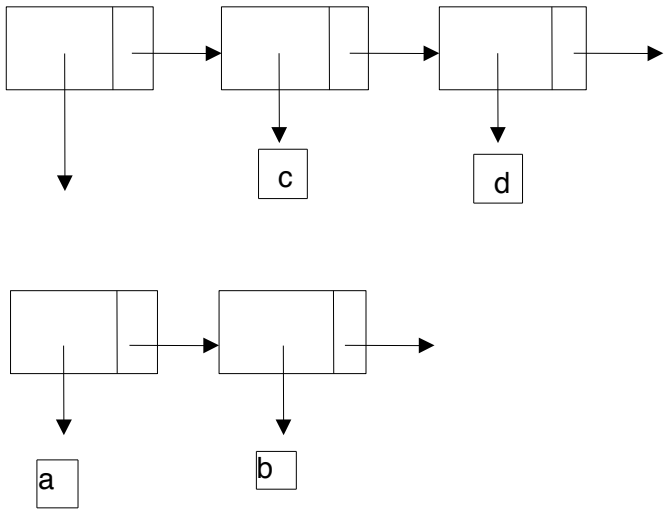

Рисунок 1 – Пример представления иерархического списка ((a b)c d)

Описание алгоритма.

Создана рекурсивная функция *explore,* принимающая в качестве аргументов указатели на два списка. Сначала выполняется проверка на конец списка. Если в одном из списков закончились элементы, а в другом еще нет, то функция возвращает *false*. Если элементы оказались атомами, то выполняется проверка на их идентичность. Если атомы различны, функция возвращает *false*.

Если элемент оказался иерархическим списком, то функция рекурсивно вызывает саму себя и производится дальнейшее сравнение.

Выполнение работы.

Для реализации программы был разработан графический интерфейс с помощью встроенного в QtCreator UI-редактора. Он представляет из себя два поля ввода, две кнопки считывания данных из файла, кнопку, запускающую синтаксический анализатор для текущих строк, поле вывода, а также кнопку выгрузки в файл информации, содержащейся в поле вывода.

Были созданы слоты, обрабатывающие сигналы clicked() кнопок. При нажатии на кнопку считывания из файла, открывается файловый диалог с помощью функции класса QFileDialog – getOpenFileName(). После выбора файла для загрузки, информация из него считывается в строку.

В функции checkLists() производится считываниедвух иерархических списков, вызов функции сравнения *explore* и вывод результата на экран.

В рекурсвиной функция bool explore((S_expr*list1, S_expr* list2, string & result)), принимающей текущие элементы списка для сравнения, производится изучение двух текущих элементов списка, если оба списка закончились и на предыдущих этапах проверки ошибок не было, функция возвращает *true, е*сли один из списков закончился, а другой нет – функция возвращает *false,* Если оба элемента списков оказались атомами, выполняется проверка на их идентичность. если атомы одинаковые, функция возвращает *true,* в противном случае – *false,* если оба элемента не атомы, то функция рекурсивно вызывает саму себя для дальнейшего анализа.

Тестирование программы.

Входные данные	Результат работы программы
() ()	These two lists are identical.
(((a b)c d) ((a b)c d)	These two lists are identical.
(a b) (a)	These two lists are different.
(e) ((a)e)	These two lists are different.

Выводы.

В ходе выполнения лабораторной работы была изучена такая структура данных как иерархические списки, а также рекурсивные методы ее обработки. Была реализована программа на C++, которая анализирует строки и определяет являются ли они идентичными иерархическими списками.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:

    void on_checkButton_clicked();

    void on_fileButtonForList1_clicked();

    void on_fileButtonForList2_clicked();

    void on_saveToFile_clicked();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Название файла: hlist.h

```
#ifndef HLIST_H
#define HLIST_H
#include <iostream>
#include <fstream>
using namespace std;
class HList;
```

```
class S_expr;
class Two_ptr{
public:
    S_expr* head;
    S_expr* tail;

    Two_ptr()
    :head{}, tail{}
    {
    }
    S_expr* getHead() const{
        return head;
    }
    S_expr* getTail() const{
        return tail;
    }
};
typedef union Node{
    char element;
    Two_ptr pair;
}Node;
class S_expr{
    bool flag;
public:
    Node node;
    S_expr()
    :flag{}, node{}
    {
    }
    bool getFlag() const{
        return flag;
    }
    Node getNode() const{
        return node;
    }
    void setFlag(bool flag){
        this->flag = flag;
    }
    void setNodeElement(char element){

        this->node.element = element;
    }
};
class HList{
    S_expr el;
public:
    HList():el{}
    {

    }
    S_expr* cons (S_expr* head, S_expr* tail, string&);
    void destroy (S_expr* s, string&);
    bool isAtom (const S_expr* s);
    bool isNull (const S_expr* s);
    S_expr* head (const S_expr* s, string&);
    S_expr* tail (const S_expr* s, string&);
    char getAtom (const S_expr* s, string&);
    S_expr* makeAtom (const char x);
    int read_s_expr(S_expr*& s, string, string&);
    int read(char prev, S_expr*& s, string &, int, string&);
```

```
    int read_seq (S_expr*& s, string &, int, string&);
    S_expr* copy_list(const S_expr* x);
};
#endif
```

Название файла: hlist.cpp

```
#include "hlist.h"
bool HList::isAtom(const S_expr* s){
    if(!s){
        return false;
    }
    return s->getFlag();
}
bool HList::isNull(const S_expr* s){
    return s==nullptr;
}
S_expr* HList::head(const S_expr* s, string & result){
    if(s){
        if(!isAtom(s)){
            return s->getNode().pair.getHead();
        }
        else{
            result.append("Error: Head(atom) \n");
            return nullptr;
        }
    }
    else{
        result.append("Error: Head(nil) \n");
        return nullptr;
    }
}
S_expr* HList::tail(const S_expr* s, string & result){
    if(s){
        if(!isAtom(s)){
            return s->getNode().pair.getTail();
        }
        else{
            result.append("Error: Head(atom) \n");
            return nullptr;
        }
    }
    else{
        result.append("Error: Head(nil) \n");
        return nullptr;
    }
}
char HList::getAtom(const S_expr* s, string & result){
    if(isAtom(s)){
        return s->getNode().element;
    }
    else{
        result.append("Error: getAtom(s) for !isAtom(s)");
        return 0;
    }
}
S_expr* HList::makeAtom(const char x){
    S_expr* s = new S_expr;
    s->setFlag(true);
    s->setNodeElement(x);
```

```
        return s;
    }
    int HList::read_s_expr(S_expr*& s, string list, string & result){
        int i=0;
        while(list[i]==' '){
            i++;
        }
        if(list[i] != '('){
            result.append("Error!!!Expression must begin with (\n");
            return 1;
        }
        if(read(list[i], s, list, i, result)){
            return 1;
        }
        return 0;
    }
    int HList::read(char prev, S_expr*& s, string & list, int i, string &
result){
        if(prev == ')') {
            result.append(" ! List.Error. List can't begin with )\n");
            return 1;
        }
        else if(prev != '('){
            s = makeAtom(prev);
        }
        else{
            if(read_seq (s, list, i, result))
                return 1;
        }
        return 0;
    }
    int HList::read_seq(S_expr*& s, string & list, int i, string & result)
{
        S_expr* p1, *p2;
        i++;
        while(list[i] == ' '){
            i++;
        }

        if(list[i] == ')'){
            s = nullptr;
        }else if(i>=list.length()){
            result.append("Error. It isn't ) in exspression\n");
            return 1;
        }
        else{
            if(read(list[i], p1, list, i, result))
                return 1;
            if(read_seq(p2, list, i, result))
                return 1;
            s = cons(p1, p2, result);
        }
        return 0;
    }
    void HList::destroy(S_expr* s, string & result){
        if(s){
            if(!isAtom(s)){
                destroy(head(s, result), result);
                destroy(tail(s, result), result);
            }
```

```
        delete s;
    }
}
S_expr* HList::cons(S_expr* head, S_expr* tail, string & result){
    S_expr* s;
    if(isAtom(tail)){
        result.append("Error tail(nil)\n");
        return nullptr;
    }
    else{
        s = new S_expr();
        s->setFlag(false);
        s->node.pair.head=head;
        s->node.pair.tail=tail;
        return s;
    }
}
```

Название файла: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "hlist.h"
#include "components.h"
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

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

bool explore(S_expr* list1, S_expr* list2, string & result){
    HList hlist;
    if(!list1 && !list2)
        return true;
    if(!list1 || !list2)
        return false;
    if(hlist.isAtom(list1) && hlist.isAtom(list2)){
        result.append("Found atoms.\n");

        result+=list1->getNode().element;
        result+='\t';
        result+=list2->getNode().element;
        result+='\n';
        if(list1->getNode().element == list2->getNode().element){
            result.append("Atoms are identical.\n");
            return true;
        }
        else
            result.append("Atoms are different.\n");
    }

    if(!hlist.isAtom(list1) && !hlist.isAtom(list2)){
        result.append("Found hierarchical lists.\n");
```

```
                        if(!explore(list1->getNode().pair.getHead(),  list2-
>getNode().pair.getHead(), result))
                return false;
            else
                    return explore(list1->getNode().pair.getTail(), list2-
>getNode().pair.getTail(), result);
        }
        return false;
    }

    void MainWindow::on_checkButton_clicked()
    {
        checkLists();
    }
    void MainWindow::checkLists(){
        HList hlist;
        S_expr* list1, *list2;
        QString input1 = ui->list1->text();
        QString input2 = ui->list2->text();
        string result;
        if(hlist.read_s_expr(list1, input1.toUtf8().constData(), result)){
            ui->resultWindow->setText(QString::fromStdString(result));
        }

            else  if(hlist.read_s_expr(list2,  input2.toUtf8().constData(),
result)){
            ui->resultWindow->setText(QString::fromStdString(result));
        }

        else if(explore(list1, list2, result)){
            result.append("These two lists are identical.");
            ui->resultWindow->setText(QString::fromStdString(result));
            hlist.destroy(list1, result);
            hlist.destroy(list2, result);
        }
        else{
            result.append("These two lists are different.\n");
            ui->resultWindow->setText(QString::fromStdString(result));
            hlist.destroy(list1, result);
            hlist.destroy(list2, result);
        }
        this->resize(482, 600);
        this->resize(481, 599);
    }
    void MainWindow::on_fileButtonForList1_clicked()
    {
        QString fileName = QFileDialog::getOpenFileName(this,
                tr("Open TXT File"),  QDir::homePath(),
                tr("TXT text (*.txt);;All Files (*)"));
        ifstream sourceFile(fileName.toUtf8().constData());
        string input;
        sourceFile >> input;
        ui->list1->setText(QString::fromStdString(input));
    }

    void MainWindow::on_fileButtonForList2_clicked()
    {
        QString fileName = QFileDialog::getOpenFileName(this,
                tr("Open TXT File"),  QDir::homePath(),
                tr("TXT text (*.txt);;All Files (*)"));
```

```
        ifstream sourceFile(fileName.toUtf8().constData());
        string input;
        sourceFile >> input;
        ui->list2->setText(QString::fromStdString(input));
    }

    void MainWindow::on_saveToFile_clicked()
    {
        QString fileName = QFileDialog::getOpenFileName(this,
                tr("Open TXT File"),  QDir::homePath(),
                tr("TXT text (*.txt);;All Files (*)"));
        ofstream sourceFile(fileName.toUtf8().constData());
                        sourceFile   <<   ui->resultWindow-
>toPlainText().toUtf8().constData();
    }
```

Название файла: components.h

```
#ifndef COMPONENTS_H
#define COMPONENTS_H
#include <QFileDialog>
#include <QMessageBox>
#include <QLabel>
#endif
```

Название файла: 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>481</width>
    <height>599</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QLineEdit" name="list1">
    <property name="geometry">
     <rect>
      <x>30</x>
      <y>60</y>
      <width>281</width>
      <height>31</height>
     </rect>
    </property>
   </widget>
   <widget class="QLineEdit" name="list2">
    <property name="geometry">
     <rect>
      <x>30</x>
      <y>100</y>
      <width>281</width>
```

```
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QLabel" name="resultLabel">
 <property name="geometry">
  <rect>
   <x>40</x>
   <y>190</y>
   <width>191</width>
   <height>21</height>
  </rect>
 </property>
 <property name="text">
  <string>Результат проверки</string>
 </property>
</widget>
<widget class="QTextEdit" name="resultWindow">
 <property name="geometry">
  <rect>
   <x>30</x>
   <y>220</y>
   <width>271</width>
   <height>331</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="checkButton">
 <property name="geometry">
  <rect>
   <x>30</x>
   <y>140</y>
   <width>113</width>
   <height>32</height>
  </rect>
 </property>
 <property name="text">
  <string>Проверить</string>
 </property>
</widget>
<widget class="QLabel" name="header">
 <property name="geometry">
  <rect>
   <x>70</x>
   <y>30</y>
   <width>391</width>
   <height>20</height>
  </rect>
 </property>
 <property name="text">
  <string>Проверить идентичность 2 ирархических списков</string>
 </property>
</widget>
<widget class="QPushButton" name="fileButtonForList1">
 <property name="geometry">
  <rect>
   <x>330</x>
   <y>60</y>
   <width>141</width>
   <height>31</height>
```

```
     </rect>
    </property>
    <property name="text">
     <string>Данные из файла</string>
    </property>
   </widget>
   <widget class="QPushButton" name="fileButtonForList2">
    <property name="geometry">
     <rect>
      <x>330</x>
      <y>100</y>
      <width>141</width>
      <height>31</height>
     </rect>
    </property>
    <property name="text">
     <string>Данные из файла</string>
    </property>
   </widget>
   <widget class="QPushButton" name="saveToFile">
    <property name="geometry">
     <rect>
      <x>322</x>
      <y>520</y>
      <width>151</width>
      <height>31</height>
     </rect>
    </property>
    <property name="text">
     <string>Сохранить в файл</string>
    </property>
   </widget>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```