**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе № 2**

**по дисциплине «Алгоритмы и Структуры Данных»**

Тема: **Иерархический список**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2017

**Цель работы.**

Знакомство с иерархическими списками. Построение алгоритмов по их реализации и обработке

**Постановка задачи.**

Заменить в иерархическом списке все вхождения заданного элемента

(атома) x на заданный элемент (атом) y.

**Организация иерархического списка.**

Каждый из элементов списка удобно представить в виде структуры данных, которая, в зависимости от своего положения, может быть либо атомом (элемент с данными), либо частью линейного списка. Описание структуры lisp представлено в приложении А.

**Ход работы.**

1. Основные функции работы с иерархическим списком помещены в отдельный класс data\_container (объявление класса и его основных функций в приложениях А и Б.):
   1. lisp head(const lisp s):

Базовая функция, возвращающая «голову списка» с предусловием, что список не пуст. const lisp s – входной параметр, указатель на голову списка.

* 1. lisp tail(const lisp s):

Базовая функция, возвращающая указатель на следующий элемент с предусловием, что список не пуст. const lisp s – входной параметр, указатель на голову списка.

* 1. lisp cons(const lisp h, const lisp t):

Базовая функция, являющаяся конструктором. const lisp h – входной параметр, указатель на голову списка. const lisp t входной параметр, указатель на хвост списка.

* 1. lisp make\_atom(const base x):

Функция формирует атом, т.е. базовый элемент списка. const base x – считываемое значение базового типа.

* 1. bool isAtom(const lisp s):

Определяет тип элемента списка (true – атом, false – список). Const lisp s – входной параметр, указатель на голову списка.

* 1. bool isNull(const lisp s):

Определяет пуст ли список (true – пуст). const lisp s – входной параметр, указатель на голову списка.

* 1. void destroy(lisp s):

Удаляет список. const lisp s – входной параметр, указатель на голову списка.

* 1. Функции считывания из строки и создания списка:
* void read\_lisp(lisp&);
* void read\_seq(lisp&);
* void read\_s\_expr(base, lisp&).
  1. Функции записи списка в строку:
* void write\_lisp(const lisp);
* void write\_seq(const lisp).
  1. Функции замены атома:
* void change\_lisp(lisp&,base,base);
* void change\_seq(lisp&,base,base);

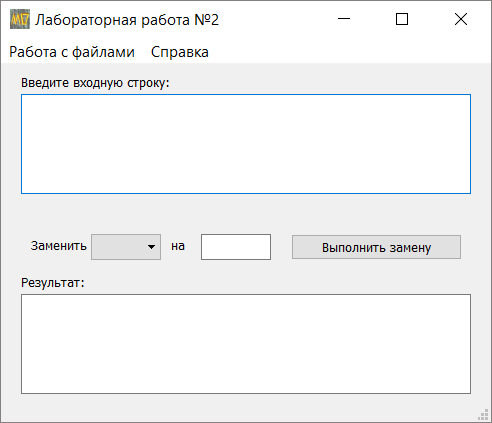
1. Программа должна считывать иерархический список из его скобочной формы записи. Также программа будет проверять исходные данные по ходу их введения. Реализация данных функций представлена в приложениях В и Г.

Рисунок 1. Вид программы

# **Пример работы программы**

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| (1 2 3 4 () )  Замена 1 -> 6 | ( 6 2 3 4 ( ) ) |
| (ee (11) )) | Введенные данные некорректны! |
| 555  Замена 555 -> 6 | 6 |
| (Pascal is (awesome) 1 )  Замена Pascal -> Qt | (Qt is (awesome) 1 ) |
| ( Qt is ( one ( of ) the ( best ) 1 ) ever )  Замена 1 -> program | ( Qt is ( one ( of ) the ( best ) program ) ever ) |

# **Вывод**

В результате выполнения данной лабораторной работы были построены функции для работы с иерархическими списками, и создана программа, использующая данные функции. Также был создан графический интерфейс для более удобного её использования.

# **ПРИЛОЖЕНИЕ А.**

# **ИХОДНЫЙ КОД MAIN.CPP**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.setWindowIcon(QIcon("C:/Downloads/2.png"));

w.show();

return a.exec();

}

# **ПРИЛОЖЕНИЕ Б.**

# **ИХОДНЫЙ КОД MAINWINDOW.CPP**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

setWindowIcon(QIcon("./24.jpg"));

//setFixedSize(457,290);

setWindowTitle("Лабораторная работа №2");

ui->outputText->setReadOnly(true);

data = new data\_container;

}

MainWindow::~***MainWindow***()

{

delete ui;

delete data;

}

void MainWindow::**generateComboBox**()

{

QString str = ui->inputText->toPlainText().simplified();

ui->comboBox->clear();

QStringList list = str.split(QRegExp("\\b"));

for (int i=0; i<list.size(); i++)

{

QString x = list[i];

if (!(x.contains("(") || x.contains(")") || x.contains(" ") ))

ui->comboBox->addItem(list[i]);

}

}

void MainWindow::**on\_inputText\_textChanged**()

{

data->inp\_str = ui->inputText->toPlainText().simplified();

if (data->brack\_analizator())

{

ui->label\_3->setText("Введенные данные корректны!");

ui->label\_3->setStyleSheet("QLabel {color : green; }");

ui->comboBox->setEnabled(true);

ui->actionButton->setEnabled(true);

ui->change\_to->setEnabled(true);

ui->label->setEnabled(true);

ui->label\_2->setEnabled(true);

print(0);

}

else

{

ui->label\_3->setText("Введенные данные некорректны! Измените входную строку.");

ui->label\_3->setStyleSheet("QLabel {color : red; }");

ui->comboBox->setDisabled(true);

ui->actionButton->setDisabled(true);

ui->change\_to->setDisabled(true);

ui->label->setDisabled(true);

ui->label\_2->setDisabled(true);

}

}

void MainWindow::**print**(bool a)

{

ui->outputText->clear();

data->out\_str.clear();

if (!a)

generateComboBox();

data->destroy(data->main\_list);

data->add\_spaces\_and\_simplifie();

data->str\_in.setString(&data->inp\_str);

data->read\_lisp(data->main\_list);

if (a)

data->change\_lisp(data->main\_list,ui->comboBox->itemText(ui->comboBox->currentIndex()).simplified(), ui->change\_to->toPlainText());

data->write\_lisp(data->main\_list);

data->out\_str = data->out\_str.simplified();

ui->outputText->setPlainText(data->out\_str);

}

void MainWindow::**on\_actionButton\_clicked**()

{

print(1);

}

void MainWindow::**on\_action\_3\_triggered**()

{

QMessageBox::about(this, tr("О программе"), tr("Данная программа заменяет в иерархическом списке \nвсе вхождения заданного элемента (атома) x на заданный элемент (атом) y"));

return;

}

void MainWindow::**on\_action\_4\_triggered**()

{

QMessageBox::about(this, tr("Об авторе"), tr("Лабораторная работа №2\n\nВыполнил: Ковынев М.В.\nГруппа: 6304"));

return;

}

void MainWindow::**on\_action\_triggered**()

{

QString FileName = QFileDialog::getOpenFileName(this, "Открыть файл...", QDir::homePath(), "Text files (\*.txt)");

if (FileName.isEmpty())

return;

QFile File(FileName);

if(File.*open*(QIODevice::ReadOnly | QIODevice::Text))

{

ui->inputText->setPlainText(File.readAll());

File.*close*();

}

else

{

QMessageBox::warning(this, "Error", QString("Не смог открыть файл %1").arg(FileName),QMessageBox::Ok);

}

}

void MainWindow::**on\_action\_2\_triggered**()

{

if(ui->outputText->toPlainText().size() == 0)

return;

QString FileName = QFileDialog::getSaveFileName(this, tr("Сохранить файл"),QDir::homePath(),

tr("Text Files (\*.txt)"));

if (FileName.isEmpty())

return;

QFile File(FileName);

if(File.*open*(QIODevice::WriteOnly | QIODevice::Text))

{

QByteArray Data;

Data.append(ui->outputText->toPlainText());

File.write(Data);

File.*close*();

}

}

# **ПРИЛОЖЕНИЕ В.**

# **ИХОДНЫЙ КОД MAINWINDOW.H**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "list.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

data\_container \*data;

private slots:

void **slotOpen**();

void **slotSave**();

void **activate\_button**();

void **make\_action**();

void **mov\_top**();

void **programm\_info**();

void **auth\_info**();

void **on\_inputText\_textChanged**();

void **on\_actionButton\_clicked**();

void **print**(bool);

void **generateComboBox**();

void **on\_action\_3\_triggered**();

void **on\_action\_4\_triggered**();

void **on\_action\_triggered**();

void **on\_action\_2\_triggered**();

};

#endif // MAINWINDOW\_H

# **ПРИЛОЖЕНИЕ Г.**

# **ИХОДНЫЙ КОД LIST.CPP**

#include "mainwindow.h"

two\_ptr::**two\_ptr**()

{

hd = NULL;

tl = NULL;

}

s\_expr::**s\_expr**()

{

tag = false;

node.atom = "";

}//end union node

data\_container::**data\_container**()

{

main\_list = NULL;

inp\_str = "";

out\_str = "";

i=0;

}

void data\_container::**add\_spaces\_and\_simplifie**()

{

int a = -2;

while((a = inp\_str.indexOf('(',a+2))!=-1)

{

inp\_str.insert(a, ' ');

inp\_str.insert(a+2, ' ');

}

a = -2;

while((a = inp\_str.indexOf(')',a+2))!=-1)

{

inp\_str.insert(a, ' ');

inp\_str.insert(a+2, ' ');

}

inp\_str = inp\_str.simplified();

}

void data\_container:: **read\_lisp**(lisp &y)

{

base x;

do str\_in >> x;

while (x==" ");

read\_s\_expr(x,y);

}

void data\_container::**read\_s\_expr**(base prev, lisp &y)

{

if(prev==")")

return;

else if (prev!="(")

y = make\_atom(prev);

else

read\_seq(y);

};

void data\_container::**read\_seq**(lisp &y)

{

base x;

lisp p1, p2;

str\_in >> x;

if((str\_in.status() == QTextStream::WriteFailed))

{

return;

}

else

{

while(x==" ")

str\_in>> x;

if(x==")")

y = NULL;

else

{

read\_s\_expr(x,p1);

read\_seq(p2);

y = cons(p1,p2);

}

}

}

lisp data\_container::**cons**(const lisp h, const lisp t)

{

lisp p;

if (isAtom(t))

{

exit(1);

}

else

{

p = new s\_expr;

if(p==NULL)

{

exit(1);

}

else

{

p->tag = false;

p->node.pair.hd = h;

p->node.pair.tl = t;

return p;

}

}

}

lisp data\_container::**make\_atom**(const base x)

{

lisp s;

s = new s\_expr;

s->tag = true;

s->node.atom = x;

return s;

}

bool data\_container::**isAtom**(const lisp s)

{

if (s==NULL)

return false;

else

return (s->tag);

}

void data\_container::**write\_lisp**(const lisp x)

{

if(isNull(x))

out\_str +=" ( )";

else if(isAtom(x))

out\_str += ' ' +x->node.atom;

else

{

out\_str+=" (";

write\_seq(x);

out\_str+= " )";

}

}

void data\_container::**write\_seq**(const lisp x)

{

if (!isNull(x))

{

write\_lisp(head(x));

write\_seq(tail(x));

}

}

lisp data\_container::**head** (const lisp s)

{

if (s != NULL)

if (!isAtom(s))

return s->node.pair.hd;

else

{

exit(1);

}

else

{

exit(1);

}

}

lisp data\_container::**tail**(const lisp s)

{

if (s != NULL)

if (!isAtom(s))

return s->node.pair.tl;

else

{

exit(1);

}

else

{

exit(1);

}

}

bool data\_container::**isNull**(const lisp s)

{

if (s == NULL)

return true;

return false;

}

void data\_container::**destroy**(lisp s)

{

if(s!=NULL)

{

if(!isAtom(s))

{

destroy(head(s));

destroy(tail(s));

}

delete s;

};

}

bool data\_container::**brack\_analizator**()

{

int count = 0;

if(inp\_str.isEmpty())

return false;

if( inp\_str.indexOf(' ') == -1

&& inp\_str.indexOf('(') == -1

&& inp\_str.indexOf(')') == -1)

return true;

if(inp\_str[0] != '(')

return false;

for (int i = 0; i < inp\_str.size(); i++)

{

if(inp\_str[i] == '(')

count++;

if(inp\_str[i] == ')')

count--;

if(count == 0 && i!= inp\_str.size()-1)

return false;

}

if (count == 0)

return true;

return false;

}

void data\_container::**change\_lisp**(lisp& x, base e, base c)

{

if(isAtom(x))

{

if(x->node.atom == e)

x->node.atom = c;

return;

}

else if(!isAtom(x))

{

change\_seq(x, e, c);

}

}

void data\_container::**change\_seq**(lisp& x, base e, base c)

{

if (!isNull(x))

{

change\_lisp(x->node.pair.hd, e,c);

change\_seq(x->node.pair.tl, e,c);

}

}

# **ПРИЛОЖЕНИЕ Д.**

# **ИХОДНЫЙ КОД LIST.H**

#include <QMainWindow>

#include <QTextStream>

#include <Qstring>

#include <QFileDialog>

#include <QMessageBox>

#include <QTextStream>

typedef QString base; // базовый тип элементов (атомов)

struct s\_expr;

struct two\_ptr {

s\_expr \*hd;

s\_expr \*tl;

two\_ptr();

} ; //end two\_ptr;

struct s\_expr {

public:

bool tag; // true: atom, false: pair

struct {

base atom;

two\_ptr pair;

} node;

s\_expr();

}; //end s\_expr

typedef s\_expr \*lisp;

class data\_container

{

public:

lisp main\_list;

QString inp\_str;

QString out\_str;

QTextStream str\_in;

QTextStream str\_out;

int i;

data\_container();

void **read\_lisp**(lisp&);

void **write\_lisp** (const lisp);

bool **brack\_analizator**();

void **destroy** (lisp);

void **add\_spaces\_and\_simplifie**();

void **change\_lisp**(lisp&, base, base);

private:

void **read\_s\_expr** (base, lisp&);

void **read\_seq** (lisp&);

lisp **head** (const lisp);

lisp **tail** (const lisp);

lisp **cons** (const lisp, const lisp);

lisp **make\_atom** (const base);

bool **isAtom** (const lisp);

bool **isNull** (const lisp);

void **write\_seq** (const lisp);

void **change\_seq**(lisp&, base, base);

};