

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 14

Студент гр. 8304

Николаева М. А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Познакомиться с нелинейной конструкцией – иерархический список, 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, для решения поставленного задания, использующую рекурсию;
- 3) сопоставить рекурсивное решение с итеративным решением задачи;
- 4) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

обратить иерархический список на всех уровнях вложения; например, для исходного списка (a (b c) d) результатом обращения будет список (d (c b) a).

Описание алгоритма.

Программа рекурсивно считывает данные, проверяет их корректность и заносит их в список. Для этого считывается очередной символ строки. Если это атом, он добавляется к иерархическому списку, если это список – рекурсивно заносится в список.

Для реализации разворота иерархического списка в цикле рекурсивно обрабатываем вложенные списки и разворачиваем указатели в обратную сторону.

Разворот списка (список) {

предыдущий элемент = нулевому указателю

следующий = нулевому указателю

```

текущий = указателю список
пока (тек. эл. не равен нулевому указателю){
    если (текущий - атом)
        вызываем рекурсию для вложенного списка
    разворачиваем указатели между предыдущим и следующим
}
список = предыдущему элементу
}

```

Спецификация программы.

Программа предназначена для разворота иерархического списка.

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 во всплывающем окне.

Тестирование.

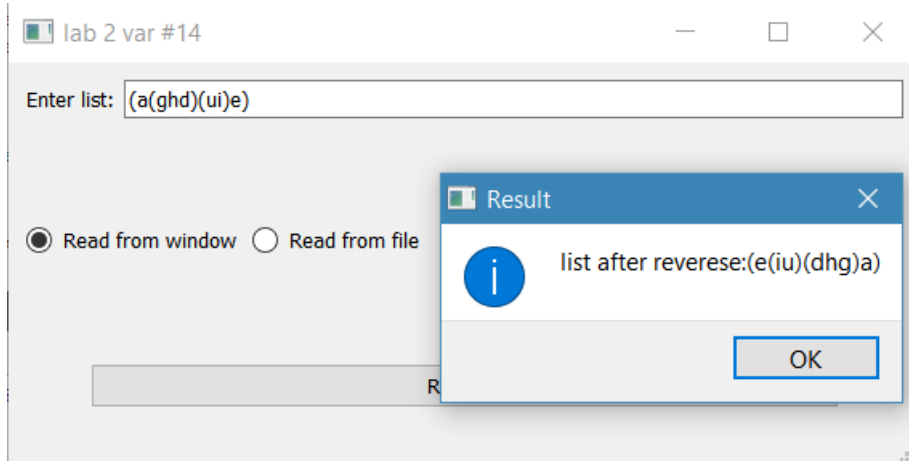


Рисунок 1- сравнение списков

Остальные тесты приведены в табл. 1.

Таблица 1 – Результаты тестирования программы

Ввод	Результат
qwe()(ad))	Некорректные данные
(Некорректные данные
(qwer(sadas)(asfas)sda()	Некорректные данные
((qwer()(vddkbbk))(ervnvn()ekvnnlkewnvkl(ifuhiuvhi)evenn)wevnjvndn(()edv)) ((vde())()ndnvjnvew(nneve(ihvuihufi)lkvnweklennvke()nvnvre)((kbbddv)()rewq))	Выполнен разворот
(weq(weq((weq)(weq)(we))egdb)egdb) (bdge(bdge((ew)(qew)(qew))qew)qew)	Выполнен разворот
(qwer) (rewq)	Выполнен Разворот
() ()	Выполнен Разворот
(((((())())))) (((())()))	Выполнен разворот

Анализ алгоритма.

Алгоритм работает за линейное время от размера списка. 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 содержит умный указатель на вложенный список (Head), умный указатель на следующий элемент списка (Tail), флаг, для определения атома, значение атома. Умные указатели были использованы во избежание утечек памяти.

Методы класса для доступа к данным.

```
ListPointer getHead() const;
ListPointer getTail() const;
bool isNull() const;
bool isAtom() const;
char getAtom() const;
```

Статический метод класса для создания иерархического списка:

```
static bool buildList(HierarchicalList::ListPointer& list, const std::string& str);
```

Принимает на вход ссылку на строку и ссылку на умный указатель на список, проверяет корректность строки и вызывает функции рекурсивного создания списка.

Метод класса для рекурсивной печати следующих списков.

```
void print_seq(QDebug& out) const;
```

Принимает на вход ссылку на выходной поток и рекурсивно выводит следующие списки.

Приватный метод класса для считывания вложенных списков:

```
static void readData(ListPointer& list, const char prev,
std::string::const_iterator& it,
const std::string::const_iterator& end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент строки и итераторы на строку. Если предыдущий элемент не равен “(“, создается атом, в ином случае вызывает считывание следующего списка.

Приватный метод класса для считывания следующих списков:

```
static void readSeq(ListPointer& list, std::string::const_iterator& it,
const std::string::const_iterator& end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент строки и итераторы на строку. Если строка пустая – происходит return. Если элемент равен “)”, создается пустой список. В ином случае рекурсивно считываются вложенные и следующие списки, затем объединяются в текущем списке.

Статический метод класса для реверсирования списка:

```
static void reverseList(ListPointer& list, size_t depth = 1);
```

Принимает на вход ссылку на указатель на список и глубину рекурсии. Для вложенного списка вызывается рекурсия, затем список разворачивается с помощью обмена указателей.

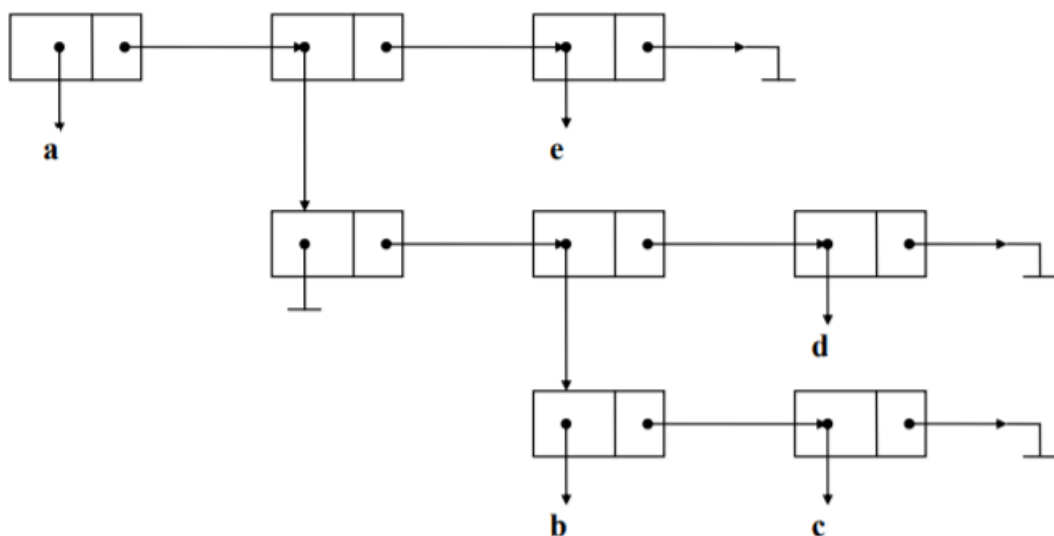
Функция для проверки корректности строки.

```
bool isCorrectStr(const std::string& str);
```

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

Рисунок 2 – Структура иерархического списка

Структура списка (a((bc)d)e)



Вывод.

В ходе выполнения данной лабораторной работы реализован класс иерархического списка и функция для реверсирования заданного списка. Научилась обрабатывать иерархические списки, с использованием базовых функций их рекурсивной обработки и сравнивать. Реализовано unit-тестирование.

Приложение А. Исходный код программы.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDebug>
#include <string>
#include <QString>
#include <QMessageBox>
#include <QTextStream>
#include <QFile>
#include <QFileDialog>
#include "hierarchicallist.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_reverse_pushButton_clicked();
    void on_readFromFile_radioButton_clicked();
    void on_readFromWindow_radioButton_clicked();

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

#endif // MAINWINDOW_H
```

hierarchicallist.h

```
#ifndef HIERARCHICALLIST_H
#define HIERARCHICALLIST_H

#define QT_NO_DEBUG_OUTPUT
```

```

#include <memory>
#include <iostream>
#include <QObject>
#include <QDebug>

class HierarchicalList : public QObject
{
    Q_OBJECT

public:
    //Умный указатель на список для избежания утечек памяти
    typedef std::shared_ptr<HierarchicalList> ListPointer;

    explicit HierarchicalList(QObject *parent = nullptr);
    ~HierarchicalList();

    HierarchicalList(const HierarchicalList&) = delete;
    HierarchicalList& operator=(const HierarchicalList&) = delete;

    //преобразование списка в строку
    std::string toStdString() const;

    //перезагрузка оператора вывода
    friend QDebug operator<< (QDebug out, ListPointer list);

public slots:
    //Функции для доступа к данным, head - указатель на вложенные списки, tail - на следующие
    ListPointer getHead() const;
    ListPointer getTail() const;
    bool isNull() const;
    bool isAtom() const;
    char getAtom() const;

    //Функция для разворота списка
    static void reverseList(ListPointer& list, size_t depth = 1);

    //Функция для создания списка из строки
    static bool buildList(HierarchicalList::ListPointer& list, const std::string& str);

    //Функция рекурсивной печати следующих списков
    void print_seq(QDebug& out) const;

private slots:
    //преобразование следующих списков в строку
    void tailToStdString(std::string& list);

    //функции рекурсивного чтения списка
    static void readData(ListPointer& list, const char prev, std::string::const_iterator& it,
        const std::string::const_iterator& end);
    static void readSeq(ListPointer& list, std::string::const_iterator& it,
        const std::string::const_iterator& end);

    //Функция конструирования непустого списка
    static ListPointer cons(ListPointer& head, ListPointer& tail);

    //превращает объект класса в атом(список с данными)
    void createAtom(const char ch);

private:
    //Флаг для определения, является ли список - атомом
    bool flag;
    //указатели на вложенный и следующий список
    ListPointer head;
    ListPointer tail;
    //данные
    char atom;
};

//Функция для проверки строки на корректность
bool isCorrectStr(const std::string& str);

#endif // HIERARCHICALLIST_H

```

main.cpp


```

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

//var #14

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("lab 2 var #14");
    w.show();

    return a.exec();
}

```

mainwindow.cpp

```

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    in = new QTextStream;
    file = new QFile;

    ui->setupUi(this);
    ui->readFromWindow_radioButton->setChecked(true);
}

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

void MainWindow::on_reverse_pushButton_clicked()
{
    qDebug() << "Start reverseList";

    HierarchicalList::ListPointer list(new HierarchicalList);
    std::string listStr = "";

    if (ui->readFromWindow_radioButton->isChecked()) {
        listStr = ui->input_lineEdit->text().toStdString();
    }
    else {
        listStr = in->readLine().toStdString();
    }

    if (HierarchicalList::buildList(list, listStr)) {
        qDebug() << "list is: " << list;
        HierarchicalList::reverseList(list);
        qDebug() << "list after reverse: " << list;
        QMessageBox::information(this, "Result", ("list after reverse:" + list-
>toStdString()).c_str());
    }
    else {
        QMessageBox::critical(this, "Result", "Wrong input data!");
        qDebug() << "input data is incorrect!";
    }

    ui->input_lineEdit->clear();
    qDebug() << "End reverseList";
}

void MainWindow::on_readFromFile_radioButton_clicked()
{
    file->close();
    ui->textLabel_0->setEnabled(true);
    ui->openFile_textLabel->setEnabled(true);
    ui->textLabel_1->setEnabled(false);
    ui->input_lineEdit->setEnabled(false);
}

```

```

        ui->input_lineEdit->clear();

        QString fileName = QFileDialog::getOpenFileName(this, "Open file", "", "*.txt");
        file->setFileName(fileName);
        file->open(QFile::ReadOnly | QFile::Text);
        in->setDevice(file);

        if (fileName != "")
            ui->openFile_textLabel->setText(fileName);
        else
            ui->openFile_textLabel->setText("none");
    }

void MainWindow::on_readFromWindow_radioButton_clicked()
{
    file->close();

    ui->textLabel_0->setEnabled(false);
    ui->openFile_textLabel->setEnabled(false);
    ui->textLabel_1->setEnabled(true);
    ui->input_lineEdit->setEnabled(true);
    ui->input_lineEdit->clear();
    ui->openFile_textLabel->setText("none");
}

```

mylist.cpp

```
#include "hierarchicallist.h"
```

```

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

    flag = false;
    atom = 0;
}

```

```

HierarchicalList::~HierarchicalList()
{
    /*
     * Т.к в классе используются умные указатели, освобождение
     * памяти происходит автоматически
     */
}

```

```

std::string HierarchicalList::toStdString() const
{
    std::string list = "";
    if (this->isNull()) {
        list += "()";
    }
    else if (this->isAtom()) {
        list += this->getAtom();
    }
    else {
        list += '(';
        list += this->head->toStdString();

        if (this->tail != nullptr)
            this->tail->tailToStdTring(list);

        list += ')';
    }

    return list;
}

```

```

void HierarchicalList::tailToStdTring(std::string &list)
{
    if (!isNull()) {
        list += this->getHead()->toStdString();
    }
}

```

```

        if (this->getTail() != nullptr)
            this->getTail()->tailToStdString(list);
    }
}

bool HierarchicalList::isNull() const
{
    /*
     * Возвращает true, если элемент является нулевым списком,
     * false - в ином случае
     */

    return (!flag && head == nullptr && tail == nullptr);
}

HierarchicalList::ListPointer HierarchicalList::getHead() const
{
    /*
     * Возвращает указатель на вложенные списки
     */
    return head;
}

HierarchicalList::ListPointer HierarchicalList::getTail() const
{
    /*
     * Возвращает указатель на следующий список
     */

    return tail;
}

bool HierarchicalList::isAtom() const
{
    /*
     * Если элемент атом - возвращает true,
     * в ином случае - false
     */

    return flag;
}

HierarchicalList::ListPointer HierarchicalList::cons(ListPointer& head, ListPointer& tail)
{
    /*
     * Функция создания списка
     */

    if (tail != nullptr && tail->isAtom()) {
        std::cerr << "Error: Tail(atom)\n";
        return nullptr;
    }
    else {
        ListPointer tmp (new HierarchicalList);
        tmp->head = head;
        tmp->tail = tail;
        return tmp;
    }
}

void HierarchicalList::print_seq(QDebug& out) const
{
    /*
     * Функция печати Tail
     */

    if (!isNull()) {
        out << this->getHead();

        if (this->getTail() != nullptr)
            this->getTail()->print_seq(out);
    }
}

```

```

}

QDebug operator<<(QDebug out, const HierarchicalList::ListPointer list)
{
    /*
     * Перегрузка оператора вывода
     */

    if (list == nullptr || list->isNull()) {
        out << "()";
    }
    else if (list->isAtom()) {
        out << list->getAtom();
    }
    else {
        out << "(";

        out << list->getHead();
        if (list->getTail() != nullptr)
            list->getTail()->print_seq(out);

        out << ")";
    }

    return out;
}

void HierarchicalList::createAtom(const char ch)
{
    /*
     * Создается объект класса - атом
     */
    this->atom = ch;
    this->flag = true;
}

void HierarchicalList::readData(ListPointer& list, const char prev, std::string::const_iterator &it,
                                const std::string::const_iterator& end)
{
    /*
     * Функция считывания данных. Считывает либо атом, либо рекурсивно считывает список
     */

    if (prev != '(') {
        list->createAtom(prev);
    }
    else {
        readSeq(list, it, end);
    }
}

void HierarchicalList::readSeq(ListPointer& list, std::string::const_iterator&it,
                                const std::string::const_iterator& end)
{
    /*
     * Функция считывания списка. Рекурсивно считывает данные и список и
     * добавляет их в исходный.
     */

    ListPointer headList(new HierarchicalList);
    ListPointer tailList(new HierarchicalList);

    if (it == end)
        return;

    if (*it == '(') {
        ++it;
    }
    else {
        char prev = *it;
        ++it;
        readData(headList, prev, it, end);
        readSeq(tailList, it, end);
    }
}

```

```

        list = cons(headList, tailList);
    }
}

bool HierarchicalList::buildList(HierarchicalList::ListPointer& list, const std::string& str)
{
    /*
     * Функция создания иерархического списка. Принимает на вход ссылку
     * на строку, проверяет корректность строки и вызывает приватный метод
     * readData().
     */

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

    if (!isCorrectStr(str))
        return false;

    auto it_begin = str.cbegin();
    auto it_end = str.cend();
    char prev = *it_begin;
    ++it_begin;
    HierarchicalList::readData(list, prev, it_begin, it_end);

    return true;
}

char HierarchicalList::getAtom() const
{
    /*
     * Функция возвращает значение атома
     */
    if (flag) {
        return atom;
    }
    else {
        qDebug() << "Error: getAtom(!atom)\n";
        return 0;
    }
}

bool isCorrectStr(const std::string &str)
{
    /*
     * 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) {
        qDebug() << "Строка не корректна!";
        return false;
    }

    if (str[0] != '(' || str[str.size() - 1] != ')') {
        qDebug() << "Строка не корректна!";
        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)) {
            qDebug() << "Строка не корректна!";
            return false;
        }
    }

    if (countBracket <= 0 && i != str.size()-1) {
        qDebug() << "Строка не корректна!";
    }
}

```

```

        return false;
    }
}

if (countBracket > 0 || i != str.size()) {
    qDebug() << "Строка некорректна!";
    return false;
}

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

void HierarchicalList::reverseList(HierarchicalList::ListPointer& list, size_t depth)
{
    /*
     * Функция, реверсирующая исходный список. Принимает на вход ссылку на
     * исходный список. Ничего не возвращает.
     *
     * В цикле реверсируется список, для вложенного списка вызывается рекурсия.
     */

    std::string debugStr = "";
    for (size_t i = 0; i < depth; ++i) {
        debugStr += " ";
    }

    qDebug() << debugStr.c_str() << "Список до разворота" << list;
    auto prev = HierarchicalList::ListPointer(nullptr);
    auto next = HierarchicalList::ListPointer(nullptr);
    auto current = list;

    while (current != nullptr && !current->isNull()) {
        //Вызов рекурсии для вложенного списка.

        if (!current->getHead()->isAtom() && !current->getHead()->isNull()) {
            reverseList(current->head, depth+1);
        }
        //Разворот списка с помощью обмена значениями указателей.

        next = current->tail;
        current->tail = prev;
        prev = current;
        current = next;
    }
    if (prev != nullptr)
        list = prev;
    qDebug() << debugStr.c_str() << "Список после разворота" << list;
}

```

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>569</width>
            <height>252</height>
        </rect>
    </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_1">
                            <property name="text">
                                <string>Enter list:</string>
                            </property>

```

```

        </widget>
    </item>
    <item>
        <widget class="QLineEdit" name="input_lineEdit"/>
    </item>
</layout>
</item>
<item>
    <layout class="QHBoxLayout" name="horizontalLayout_3">
        <item>
            <widget class="QRadioButton" name="readFromWindow_radioButton">
                <property name="text">
                    <string>Read from window</string>
                </property>
            </widget>
        </item>
        <item>
            <widget class="QRadioButton" name="readFromFile_radioButton">
                <property name="text">
                    <string>Read from file</string>
                </property>
            </widget>
        </item>
        <item>
            <spacer name="horizontalSpacer_3">
                <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="QLabel" name="textLabel_0">
                <property name="text">
                    <string>Open file:</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="reverse_pushButton">
                <property name="text">
                    <string>Reverse</string>
                </property>
            </widget>
        </item>
    </layout>
</item>

```

```

</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>
</layout>
</widget>
<widget class="QMenuBar" name="menuBar">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>569</width>
      <height>26</height>
    </rect>
  </property>
</widget>
<widget class="QStatusBar" name="statusBar"/>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>

```

Приложение В. Unit-тестирование.

tst_test_hierarchicallist.cpp

```

#include <QtTest>
#include <QCoreApplication>
#include "../lab2/hierarchicallist.h"

Q_DECLARE_METATYPE(HierarchicalList::ListPointer);
Q_DECLARE_METATYPE(HierarchicalList::ListPointer *);

class Test_HierarchicalList : public QObject
{
    Q_OBJECT

public:
    Test_HierarchicalList() = default;
    ~Test_HierarchicalList() = default;

private slots:
    void initTestCase();
    void cleanupTestCase();
    void buildListTest_data();
    void buildListTest();
    void reverseListTest_data();
    void reverseListTest();

private:
    HierarchicalList::ListPointer* list_1;
    HierarchicalList::ListPointer* list_2;
    HierarchicalList::ListPointer* list_3;
    HierarchicalList::ListPointer* list_4;
    HierarchicalList::ListPointer* list_5;
    HierarchicalList::ListPointer* list_6;

```



```

        HierarchicalList::ListPointer* list_7;
};

void Test_HierarchicalList::initTestCase()
{
    list_1 = new HierarchicalList::ListPointer(new HierarchicalList);
    list_2 = new HierarchicalList::ListPointer(new HierarchicalList);
    list_3 = new HierarchicalList::ListPointer(new HierarchicalList);
    list_4 = new HierarchicalList::ListPointer(new HierarchicalList);
    list_5 = new HierarchicalList::ListPointer(new HierarchicalList);
    list_6 = new HierarchicalList::ListPointer(new HierarchicalList);
    list_7 = new HierarchicalList::ListPointer(new HierarchicalList);
}

void Test_HierarchicalList::buildListTest_data()
{
    QTest::addColumn<QString>("input");
    QTest::addColumn<HierarchicalList::ListPointer>("list");
    QTest::addColumn<bool>("result");

    QTest::newRow("testInit_1") << "qwe() (ad)" << *list_1 << false;
    QTest::newRow("testInit_2") << "(" << *list_2 << false;
    QTest::newRow("testInit_3") << ")" << *list_3 << false;
    QTest::newRow("testInit_4") << "(qwer(sadas)(asfas)sda()" << *list_4 << false;
    QTest::newRow("testInit_5") << "(abc)" << *list_5 << true;
    QTest::newRow("testInit_6") << "(abc(asd)asd)" << *list_6 << true;
    QTest::newRow("testInit_7") << "(abc() (asd)sad)" << *list_7 << true;
}

void Test_HierarchicalList::buildListTest()
{
    QFETCH(QString, input);
    QFETCH(bool, result);
    QFETCH(HierarchicalList::ListPointer, list);

    QCOMPARE(HierarchicalList::buildList(list, input.toStdString()), result);
}

void Test_HierarchicalList::reverseListTest_data()
{
    QTest::addColumn<HierarchicalList::ListPointer>("list");
    QTest::addColumn<QString>("result");

    HierarchicalList::buildList(*list_1,
"( (qwer() (vddkbbk) ) (ervnnvn() ekvnnlkewnkvkl (ifuhiuvhi) evenn) wevnjvndn() () edv) )");
    HierarchicalList::buildList(*list_2, "(weq(weq((weq) (weq) (we) ) egdb) egdb)");
    HierarchicalList::buildList(*list_3, "(qwer)");
    HierarchicalList::buildList(*list_4, "()");
    HierarchicalList::buildList(*list_5, "(((((((())))))));

    HierarchicalList::reverseList(*list_1);
    HierarchicalList::reverseList(*list_2);
    HierarchicalList::reverseList(*list_3);
    HierarchicalList::reverseList(*list_4);
    HierarchicalList::reverseList(*list_5);

    QTest::newRow("testReverse_1") << *list_1 <<
"( (vde() () ) ndnvjnvew (nneve (ihvuihufi) lkvnweklennvke() nvnnvre) ((kbkddv) () rewq) )";
    QTest::newRow("testReverse_2") << *list_2 << "(bdge (bdge ((ew) (qew) (qew) ) qew) qew)";
    QTest::newRow("testReverse_3") << *list_3 << "(rewq)";
    QTest::newRow("testReverse_4") << *list_4 << "()";
    QTest::newRow("testReverse_5") << *list_5 << "(((((((())))))));
}

void Test_HierarchicalList::reverseListTest()
{
    QFETCH(QString, result);
    QFETCH(HierarchicalList::ListPointer, list);

    QCOMPARE(list->toStdString(), result.toStdString());
}

```

```
void Test_HierarchicalList::cleanupTestCase()
{
    delete list_1;
    delete list_2;
    delete list_3;
    delete list_4;
    delete list_5;
    delete list_6;
    delete list_7;
}

QTEST_MAIN(Test_HierarchicalList)

#include "tst_test_hierarchicallist.moc"
```