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 11

Студент гр. 8304 Масалыкин Д.Р.

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.

Сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с сокращённой скобочной записью исходного иерархического списка после устранения всех внутренних скобок;

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 создания линейного списка программа проходит по иерархическому списку и спускается на уровень ниже если встречает такой переход. После прохода по нижнему уровню программа возвращается к прежней позиции.

создание_линейного_списка(список_и, список_л) {

если атом

вставка_назад

иначе

переход на нижний уровень

повторять_пока(хвост_не_пуст)

Спецификация программы.

Программа предназначена для создания линейного списка на основе иерархического.

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

Тестирование.

```
Создание линейного списка:
Проверка на корректность: (A(B(C)D)E)
Строка корректна.
Строка корректна, иерархический список создан
__Считанный спискок__:
Список: ( A ( B ( C ) D ) E )
(ABCDE)

Создание линейного списка:
Проверка на корректность: (A(B(C)D)E)
Строка корректна.
Строка корректна, иерархический список создан
__Считанный спискок__:
Список: ( A ( B ( C ) D ) E )
(ABCDE)
```

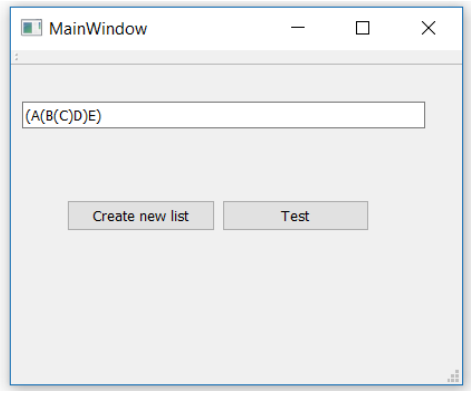


Рисунок 1- сравнение списков

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()(vddkbk))(ervnvn()ekvnnlkewnvkl(ifuhiuvhi)evenn)wevnjvndn(()()edv)) ((vde()())ndnvjnvew(nneve(ihvuihufi)lkvnweklnnvke()nvnvre)((kbkddv)()rewq))	Список создан
(weq(weq((weq)(weq)(we))egdb)egdb) (bdge(bdge((ew)(qew)(qew))qew)qew)	Список создан
(qwer)	Список создан
() ()	Список создан
(((((()))))) (((((())))))	Список создан

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:

```
void create_new_list(MyList::MyListP inp_list, std::list<char>* new_list, int depth);
```

Принимает на вход ссылку на указатель на список, указатель на линейный список и глубину рекурсии. Для вложенного списка вызывается рекурсия, затем создается линейный список.

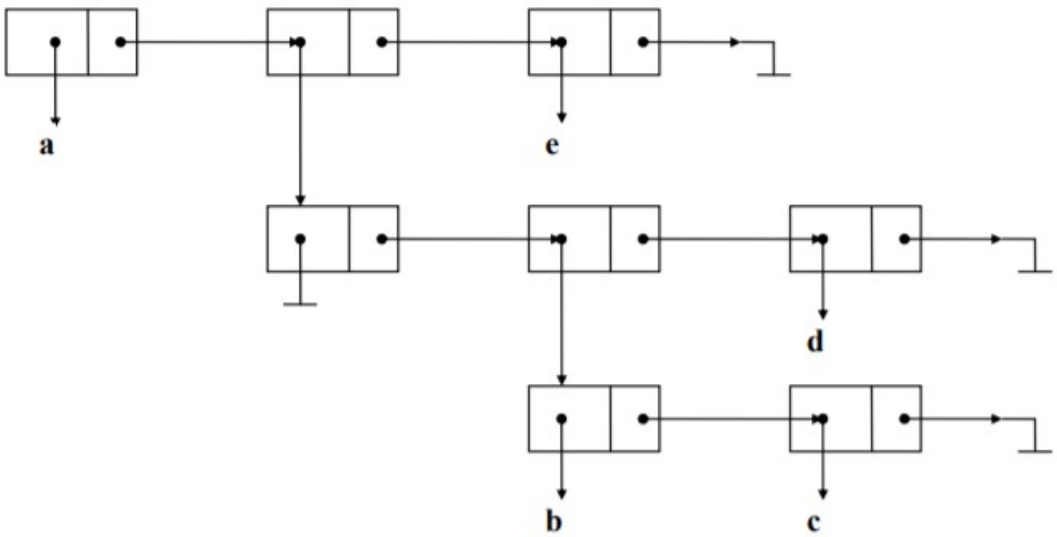
Функция для проверки корректности строки.

```
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 писка. 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 <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();
    void create_new_list(MyList::MyListP inp_list, std::list<char>* new_list, int depth);
    void print_list(std::list<char> list);

private slots:
    void on_create_pushButton_clicked();

    void on_test_pushButton_clicked();
```

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

#endif // MAINWINDOW_H
```

MyList.h

```
#ifndef MYLIST_H
#define MYLIST_H

#include <QObject>
#include <QDebug>
#include <vector>
#include <string>
#include <iostream>
#include <memory>
#include <list>

class MyList : public QObject
{

    /*
     * Класс для работы с иерархическими списками
    */
    Q_OBJECT
public:
    typedef std::shared_ptr<MyList> MyListP;

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

    MyList& operator=(const MyList& list) = delete;
    MyList(const MyList& list) = delete;

    MyListP getHead() const;
    MyListP getTail() const;
    bool isNull() const;
    bool getIsAtom() const;
    char getAtom() const;

    //void create_new_list(MyListP inp_list, std::list<char>& new_list);
    static bool buildList(MyListP& list, const std::string& str);
    friend QDebug operator<< (QDebug out, const MyListP list);

private:
    static bool checkStr(const std::string& str);
    static void readData(MyListP& list, const char prev, std::string::const_iterator& it,
                         const std::string::const_iterator& end);
    static void readSeq(MyListP& list, std::string::const_iterator& it,
                        const std::string::const_iterator& end);
    static MyListP cons(MyListP& head, MyListP& tail);
    void createAtom(const char ch);
    void print_seq(QDebug& out) const;

private:
    bool isAtom;
    MyListP head;
    MyListP tail;
    char atom;
};

#endif // MYLIST_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)
{
    ui->setupUi(this);
}

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

void MainWindow::on_create_pushButton_clicked()
{
    qDebug();
    qDebug() << "Создание линейного списка:";

    std::string inpStr = "";
    inpStr = ui->lineEdit->text().toStdString();

        //создание списков, проверка на корректность
    MyList::MyListP List(new MyList);

    if (MyList::buildList(List, inpStr)) {
        qDebug() << "Строка корректна, иерархический список создан";
        }
    else {
            qDebug() << "Строка некорректна!";
            QMessageBox::critical(this, "Result", "Входные данные неккоректны.");
            return;
        }

        qDebug() << "__Считанный спискок__:";
        qDebug() << "Список:" << List;
        std::list<char> new_list;
        new_list = {};
        create_new_list(List, &new_list, 1);
        print_list(new_list);
}

void MainWindow::create_new_list(MyList::MyListP inp_list, std::list<char>* new_list, int depth){
    depth++;
    if(inp_list->isNull())
        return;
    while(!inp_list->isNull()){
```

```
        if(inp_list->getHead()->getIsAtom()){
            new_list->push_back(inp_list->getHead()->getAtom());
            inp_list = inp_list->getTail();
        }
        else {
            create_new_list(inp_list->getHead(), new_list, depth);
            inp_list = inp_list->getTail();
        }
    }
}

void MainWindow::print_list(std::list<char> list){
    auto begin = list.begin();
    auto end = list.end();
    std::cout<<"(";
    for(auto i = list.begin(); i != list.end(); i++){
       std::cout <<*i;
    }
    std::cout <<")"<<std::endl;
}

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

    qDebug() << "Тестирование.";
    std::string firstStr = "";
    QFile file(QApplication::applicationDirPath() + "/Tests/test_correct.txt");
    file.open(QIODevice::ReadOnly);

        qDebug();

        while (!file.atEnd()) {
            firstStr = file.readLine().toStdString();
            firstStr[firstStr.size() - 1] = '\0';
            MyList::MyListP firstList(new MyList);

            if (MyList::buildList(firstList, firstStr)) {
                qDebug() << "Строка корректна, список создан";
            }
            else {
                qDebug() << "Список создан";
                continue;
            }

            qDebug() << "Список:" << firstList;

            qDebug();
        }
        std::string secondStr = "";
        QFile file2(QApplication::applicationDirPath() + "/Tests/test_incorrect.txt");
        file2.open(QIODevice::ReadOnly);

            qDebug();

            while (!file2.atEnd()) {
                secondStr = file2.readLine().toStdString();
                secondStr[secondStr.size() - 1] = '\0';
                MyList::MyListP secondList(new MyList);

                if (MyList::buildList(secondList, secondStr)) {
                    qDebug() << "Строка корректна, список создан";
                }
                else {
                    qDebug() << "Строка некорректна!";
                    continue;
                }
```

```
            qDebug() << "Список:" << secondList;

            qDebug();
        }
}
```

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, освбождение
     * памяти происхоит 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()->tailToStdTring(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
{
    /*
     * Возврщает указатель на следующий список
    */

    return tail;
}


bool HierarchicalList::isAtom() const
{
    /*
     * Если элемент атом - возращает 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, лио 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. Рекусривно считывает данные и спискок 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 неккоректна!";
        return false;
    }

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


void HierarchicalList::reverseList(HierarchicalList::ListPointer& list, size_t depth)
{
    /*
```

```
     * Функция, реверсирующая исходный список. Принмиает 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>
  <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>
```