Министерство образования Республики Беларусь

Управление образования

«Белорусский государственный университет информатики и радиоэлектроники»

Отчёт

По лабораторной работе №6

«**Наследование и механизм виртуальных функций.**

**Клиент-серверное приложение Qt**  »

|  |  |
| --- | --- |
|  | Выполнил:  студент группы 262101  Калинков Даниил Русланович  Проверил:  Ловецкий Михаил Юрьевич |

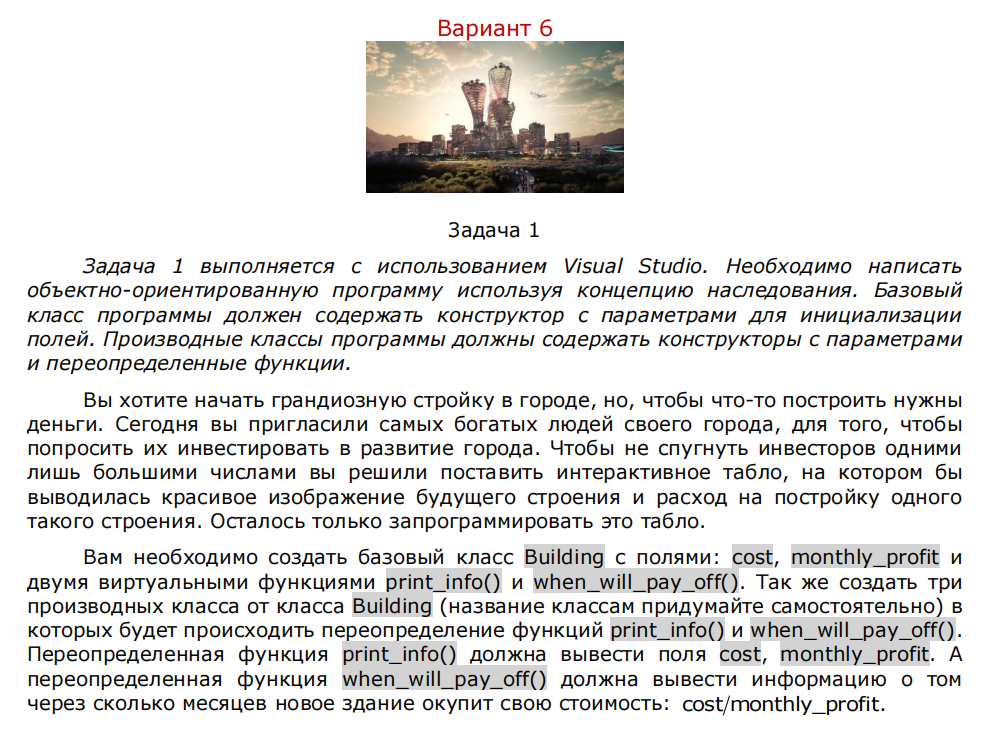
Минск, 2023

*Цель работы:* Изучить концепцию наследования и механизм виртуальных

функций. Научиться создавать клиент-серверное приложение Qt.

Вариант 6

Задание 1:



Реализация:

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

class Building

{

protected:

int cost;

int monthly\_profit;

public:

Building(int initial\_cost, int initial\_monthly\_profit)

: cost(initial\_cost), monthly\_profit(initial\_monthly\_profit) {}

virtual void print\_info() {

cout << "Building Cost: $" << cost << endl;

}

virtual void when\_will\_pay\_off() {

cout << "Monthly Profit: $" << monthly\_profit << endl;

}

};

class Home\_Buildings : public Building

{

public:

Home\_Buildings()

: Building(rand() % 100000, rand() % 100000) {}

void print\_info() {

cout << "Home Building Cost: $" << cost << endl;

cout << "Home Monthly Profit: $" << monthly\_profit << endl;

}

void when\_will\_pay\_off() {

cout << "How many months will it take for the new building to pay for itself? cost monthly\_profit:15 " << endl;

}

};

class Office\_Buildings : public Building

{

public:

Office\_Buildings()

: Building(rand() % 100000, rand() % 100000) {}

void print\_info() {

cout << "Office Building Cost: $" << cost << endl;

cout << "Office Monthly Profit: $" << monthly\_profit << endl;

}

void when\_will\_pay\_off() {

cout << "How many months will it take for the new building to pay for itself? cost monthly\_profit:14 " << endl;

}

};

class School\_Buildings : public Building

{

public:

School\_Buildings()

: Building(rand() % 100000, rand() % 100000) {}

void print\_info() {

cout << "School Building Cost: $" << cost << endl;

cout << "School Monthly Profit: $" << monthly\_profit << endl;

}

void when\_will\_pay\_off() {

cout << "How many months will it take for the new building to pay for itself? cost monthly\_profit:5 " << endl;

}

};

int main()

{

srand(static\_cast<unsigned>(time(nullptr)));

Home\_Buildings home;

Office\_Buildings office;

School\_Buildings school;

home.print\_info();

home.when\_will\_pay\_off();

office.print\_info();

office.when\_will\_pay\_off();

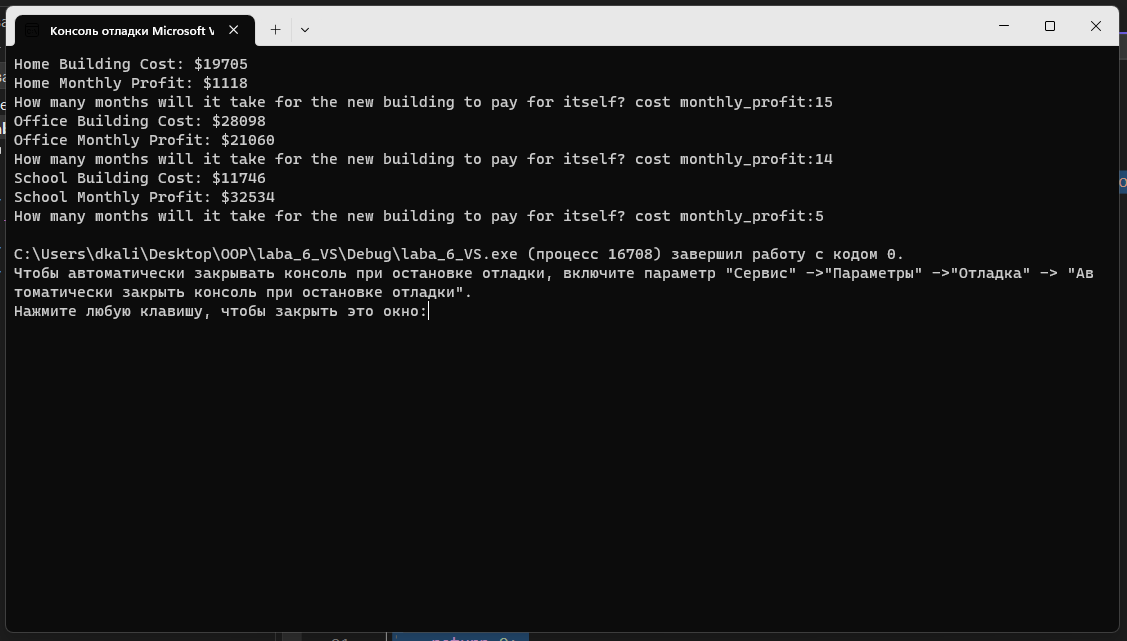
school.print\_info();

school.when\_will\_pay\_off();

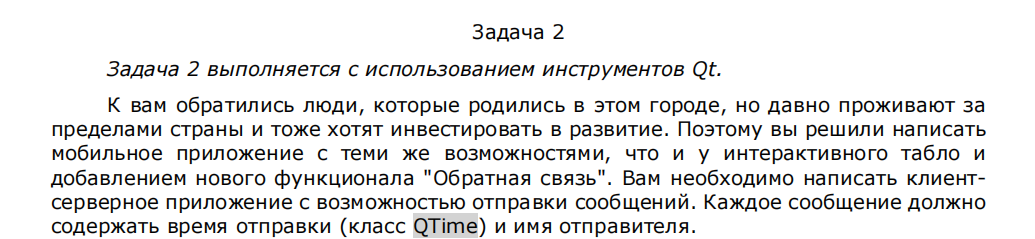
return 0;

}

результат:



Задание 2:



Задание выпонено:

**server.h:**

#ifndef SERVER\_H

#define SERVER\_H

#include <QTcpServer>

#include <QTcpSocket>

#include <QDataStream>

#include <QVector>

#include <QTime>

class **Server** : public QTcpServer

{

Q\_OBJECT

public:

**Server**();

QTcpSocket \*socket;

private:

QVector <QTcpSocket\*> Sockets;

QByteArray Data;

void **SendToClient**(QString str);

quint16 nextBlockSize;

public slots:

void ***incomingConnection***(qintptr socketDescriptor);

void **slotReadyRead**();

};

#endif // SERVER\_H

**server.cpp:**

#include "server.h"

#include <QTcpSocket>

#include <QDataStream>

Server::**Server**()

{

if(this->listen(QHostAddress::Any,2323))

{

qDebug()<<"Start";

}

else

{

qDebug()<<"error";

}

nextBlockSize=0;

}

void Server::***incomingConnection***(qintptr socketDescriptor)

{

socket = new QTcpSocket;

socket->*setSocketDescriptor*(socketDescriptor);

connect(socket, &QTcpSocket::readyRead,this, &Server::slotReadyRead);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

Sockets.push\_back(socket);

qDebug() << "client connected" << socketDescriptor;

}

void Server::**slotReadyRead**()

{

socket = (QTcpSocket\*)sender();

QDataStream in(*socket*);

in.setVersion(QDataStream::Qt\_5\_0);

if(in.status() == QDataStream::Ok)

{

qDebug()<<"nextBlockSize = 0";

//QString str;

//in>>str;

//qDebug()<<"str";

//SendToClient(str);

for(;;)

{

if(nextBlockSize == 0)

{

if(socket->*bytesAvailable*()<2)

{

qDebug()<<"Data < 2, break";

break;

}

in>>nextBlockSize;

qDebug()<<"nextBlockSize = "<<nextBlockSize;

}

if(socket->*bytesAvailable*()<nextBlockSize)

{

qDebug()<<"Data not full Break";

break;

}

QString str;

QTime time;

in >>time>>str;

nextBlockSize=0;

qDebug()<<str;

SendToClient(str);

break;

}

}

else

{

qDebug()<<"DataStream error";

}

}

void Server::**SendToClient**(QString str)

{

Data.clear();

QDataStream out(*&Data*, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_9);

out<<quint16(0)<<QTime::currentTime()<<str;

out.device()->*seek*(0);

out<<quint16(Data.size() - sizeof(quint16));

//socket->write(Data);

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

{

Sockets[i]->write(Data);

}

}

**client.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTcpSocket>

#include <QTime>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

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

void **on\_pushButton\_2\_clicked**();

void **on\_lineEdit\_returnPressed**();

private:

Ui::MainWindow \*ui;

QTcpSocket \*socket;

QByteArray Data;

void **SendToServer**(QString str);

quint16 nextBlockSize;

public slots:

void **slotReadyRead**();

};

#endif // MAINWINDOW\_H

**client.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

socket = new QTcpSocket(this);

connect(socket, &QTcpSocket::readyRead,this, &MainWindow::slotReadyRead);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

nextBlockSize=0;

}

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

{

delete ui;

}

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

{

socket->*connectToHost*("127.0.0.1",2323);

}

void MainWindow::**SendToServer**(QString str)

{

Data.clear();

QDataStream out(*&Data*, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_9);

out<<quint16(0)<<QTime::currentTime()<<str;

out.device()->*seek*(0);

out<<quint16(Data.size() - sizeof(quint16));

socket->write(Data);

ui->lineEdit->clear();

}

void MainWindow::**slotReadyRead**()

{

QDataStream in(*socket*);

in.setVersion(QDataStream::Qt\_5\_9);

if(in.status()==QDataStream::Ok)

{

//QString str;

// in>>str;

// ui->textBrowser->append(str);

for(;;)

{

if(nextBlockSize == 0)

{

if(socket->*bytesAvailable*()<2)

{

break;

}

in>>nextBlockSize;

}

if(socket->*bytesAvailable*()<nextBlockSize)

{

break;

}

QString str;

QTime time;

in>>time>>str;

in>>str;

nextBlockSize=0;

ui->textBrowser->append(time.toString() + str);

}

}

else

{

ui->textBrowser->append("read error");

}

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

SendToServer(ui->lineEdit->text());

}

void MainWindow::**on\_lineEdit\_returnPressed**()

{

SendToServer(ui->lineEdit->text());

}

**Оно работает:**

