Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення

**Звіт**

до лабораторних робіт №1-4

на тему «Середовище програмування»

з дисципліни “Об’єктно-орієнтоване програмування”

**Лектор:**

доцент кафедри ПЗ

Коротєєва Т.О.

**Виконав:**

студ. групи ПЗ-11

Ясногородський Н.В.

**Прийняв:**

доц. Коротєєва Т.О.

«\_\_\_» \_\_\_\_\_\_ 2022 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів 2022

**Лабораторна №1**

**Тема.**Ознайомлення із середовищем розробки Borland C++ Builder 6. Створення проекту та налаштування його властивостей.

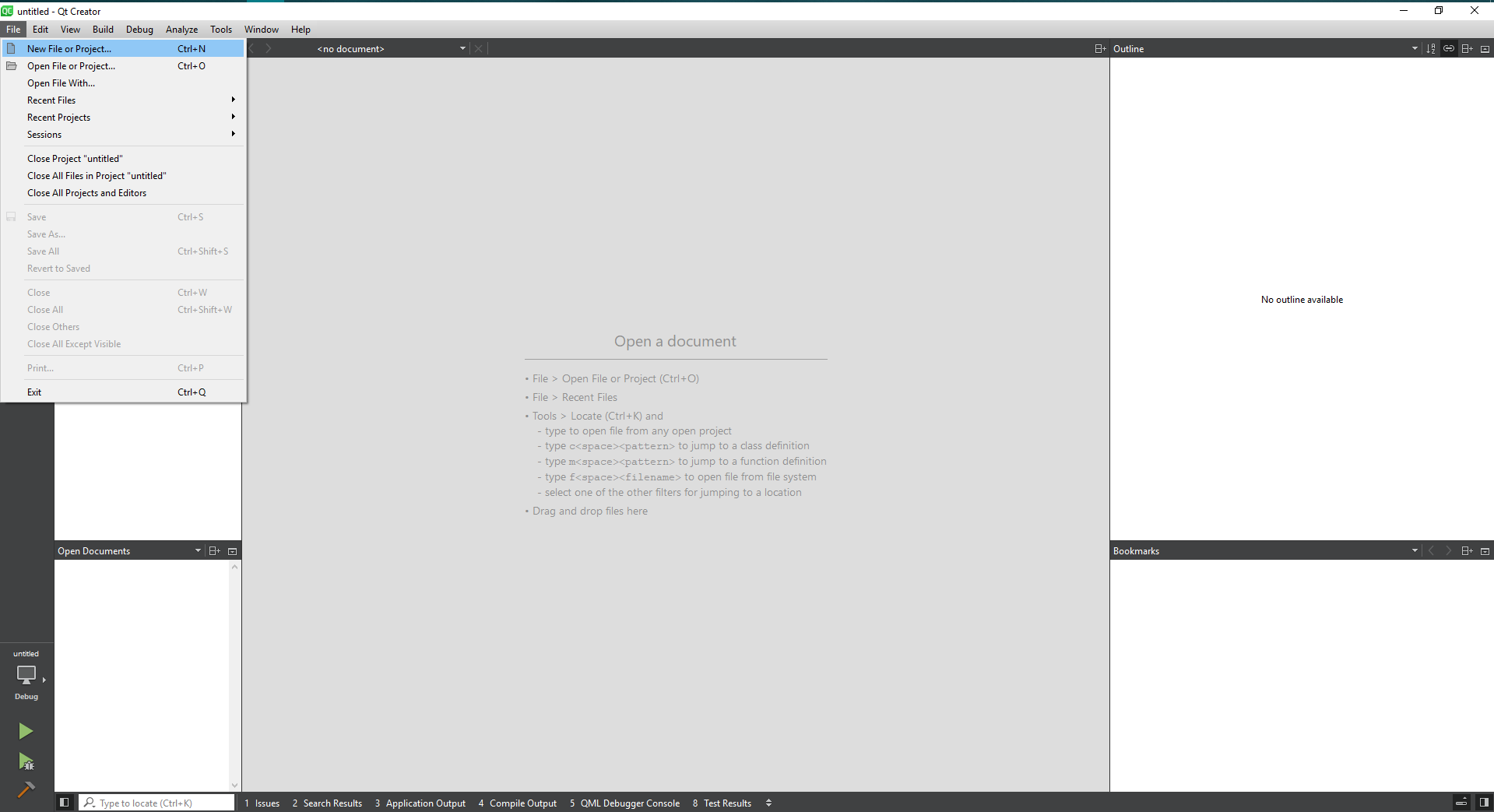
**Мета.** Засвоїти принцип візуального програмування шляхом створення та налаштування проекту.

**Теоретичні відомості.**

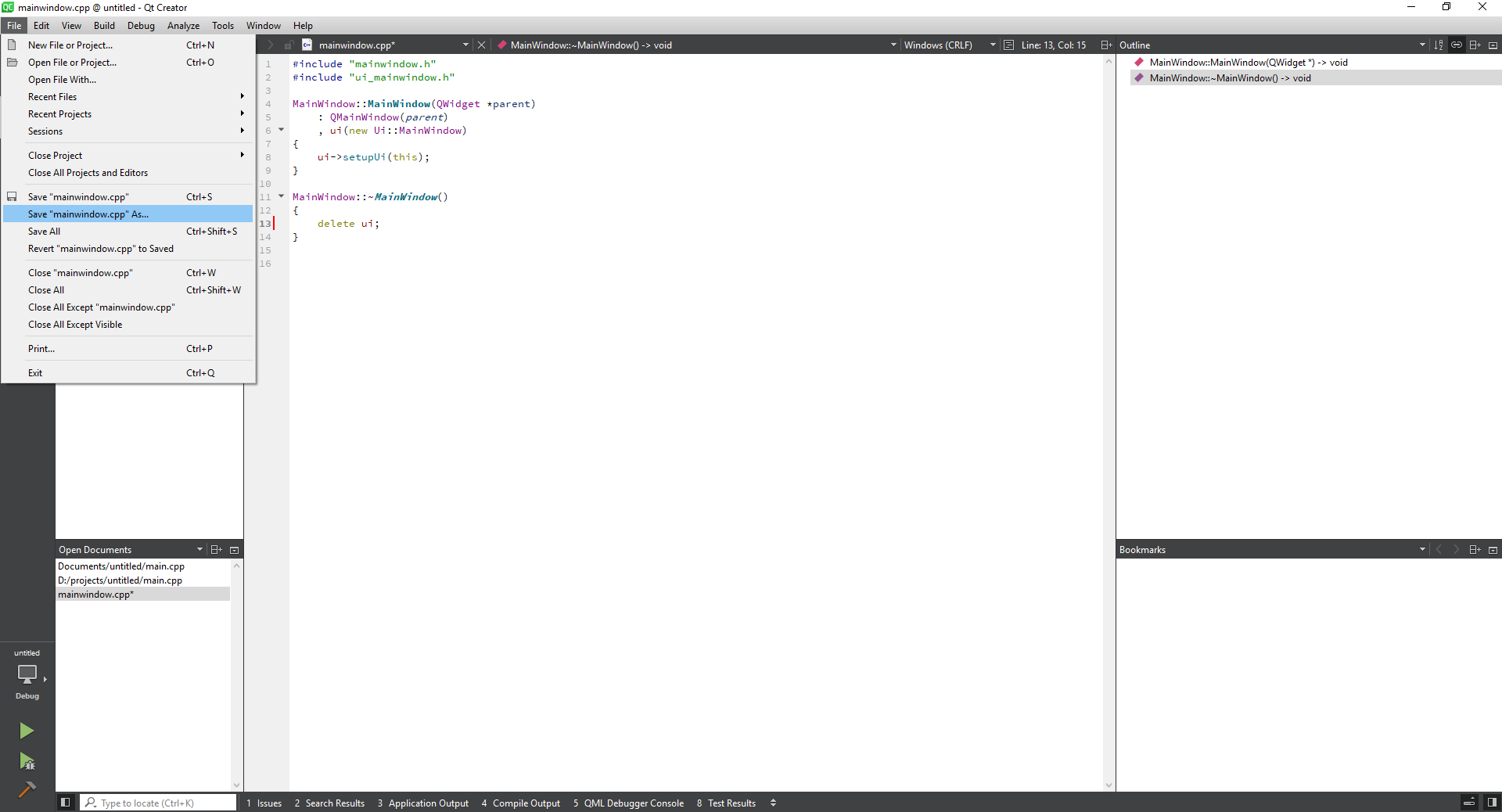
Інтерфейс **Borland C++ Builder**називають середовищем швидкої розробки застосувань **RAD (Rapid Application Development**) або середовищем візуальної розробки. Таку назву цей інтерфейс отримав за те, що створення застосування в ньому зводиться в основному до простого конструювання вікна майбутнього застосування із набору готових компонент, а більшу частину стандартних операцій виконує комп’ютер.

Завдання для лабораторної роботи

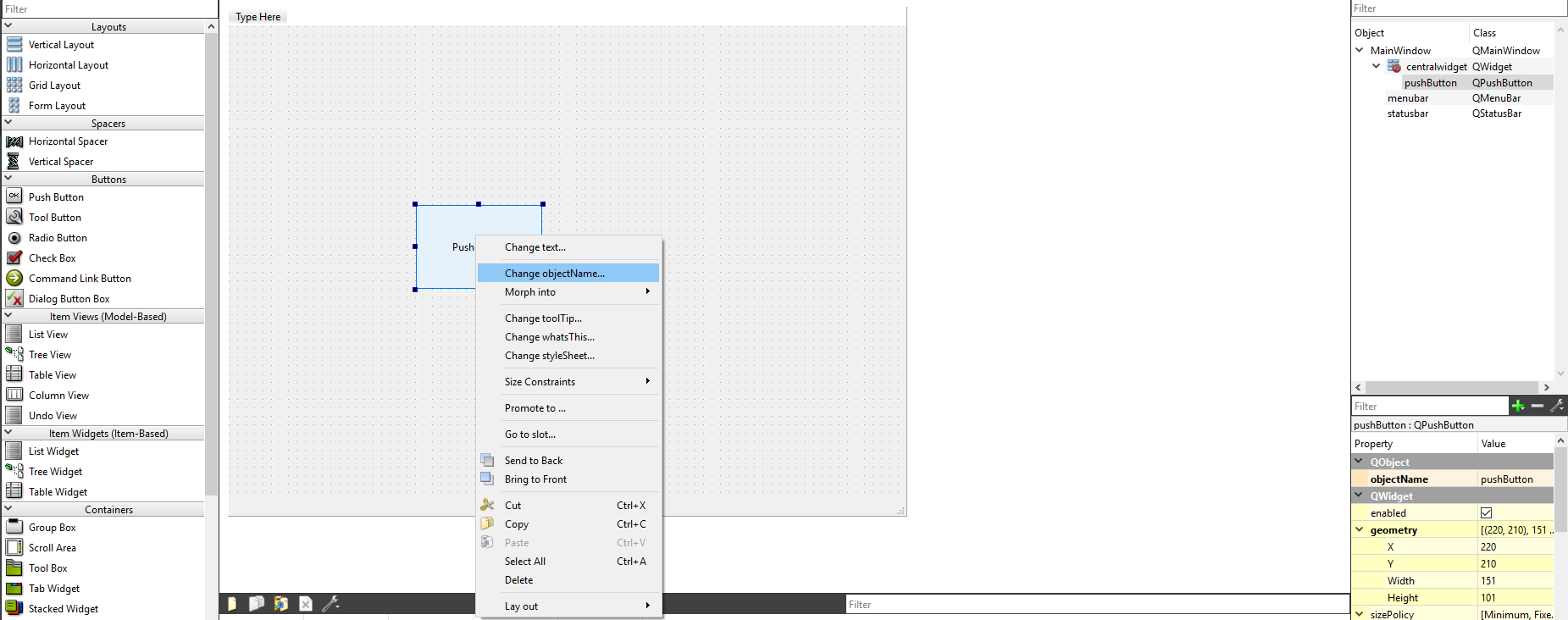
1. Ознайомитись із середовищем **Borland C++ Builder 6**.
2. Створити новий проект. Зберегти його двома способами – через  
   комбінації швидких клавіш та через меню.
3. Проглянути у вікні інспектора об’єктів  
   властивості форми. Змінити назву форми та її розміри.
4. Запустити на виконання застосування.
5. Відкрити опції проекту, змінити  
   налаштування на закладках **Application, Compiler, Packages.**Запустити на виконання застосування.



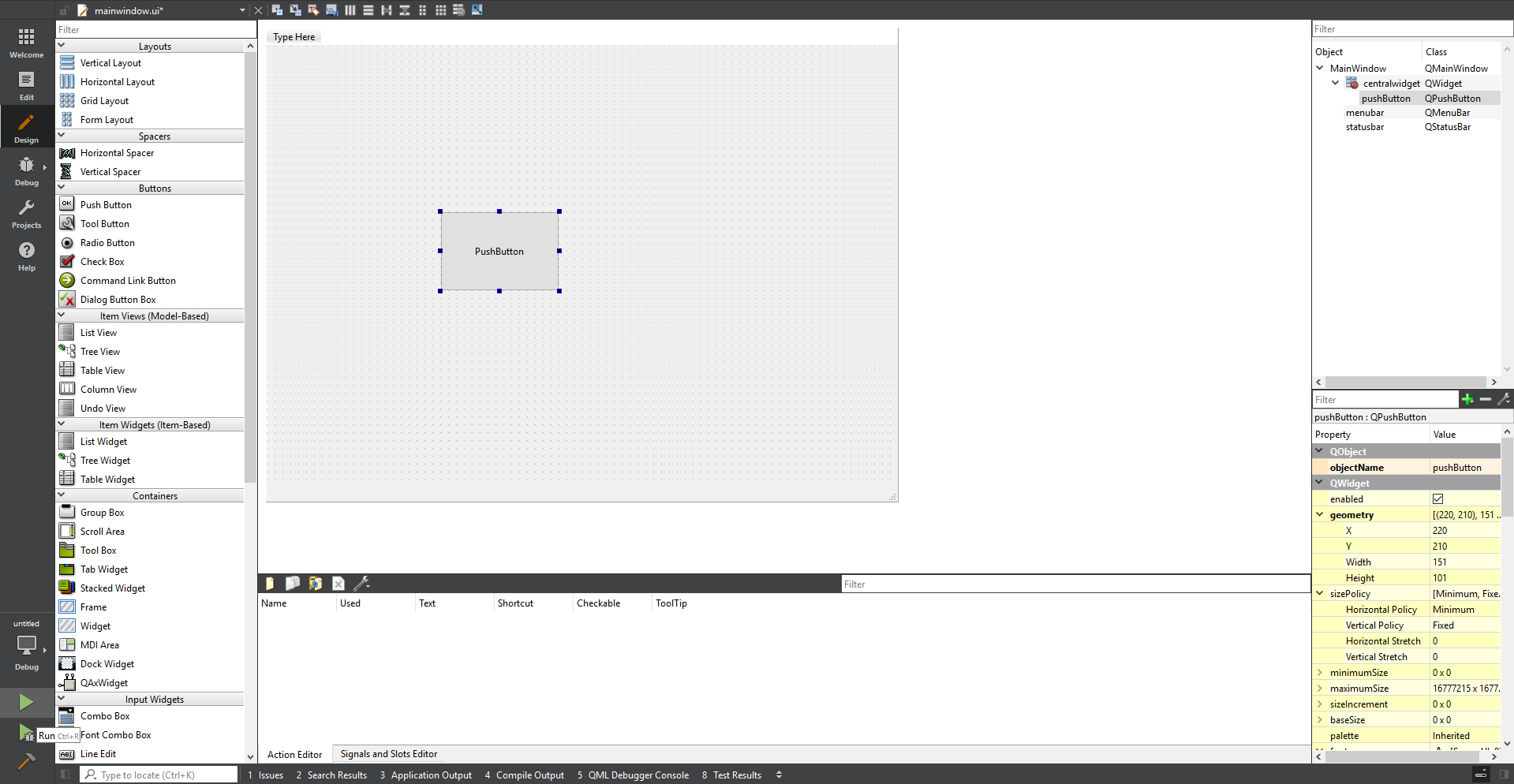
1.Створення нового проекта



2.Збереження через меню або комбінацією клавіш Ctrl+S



3.Вікно інспектора та зміна форми і назви «PushButton»



4. Запуск програми Ctrl+R або клік на зелений трикутник

Висновок: Я ознайомився з фреймворком “QT” та створив свою першу програму з кнопкою. Дізнався про особливості даного редактора.

**Лабораторна №2**

**Тема.**Базові візуальні компоненти **Borland C++ Builder 6**. Створення проекту із використанням візуальних компонент.

**Мета.** Створити віконний проект та продемонструвати використання візуальних компонент **Borland C++ Builder**.

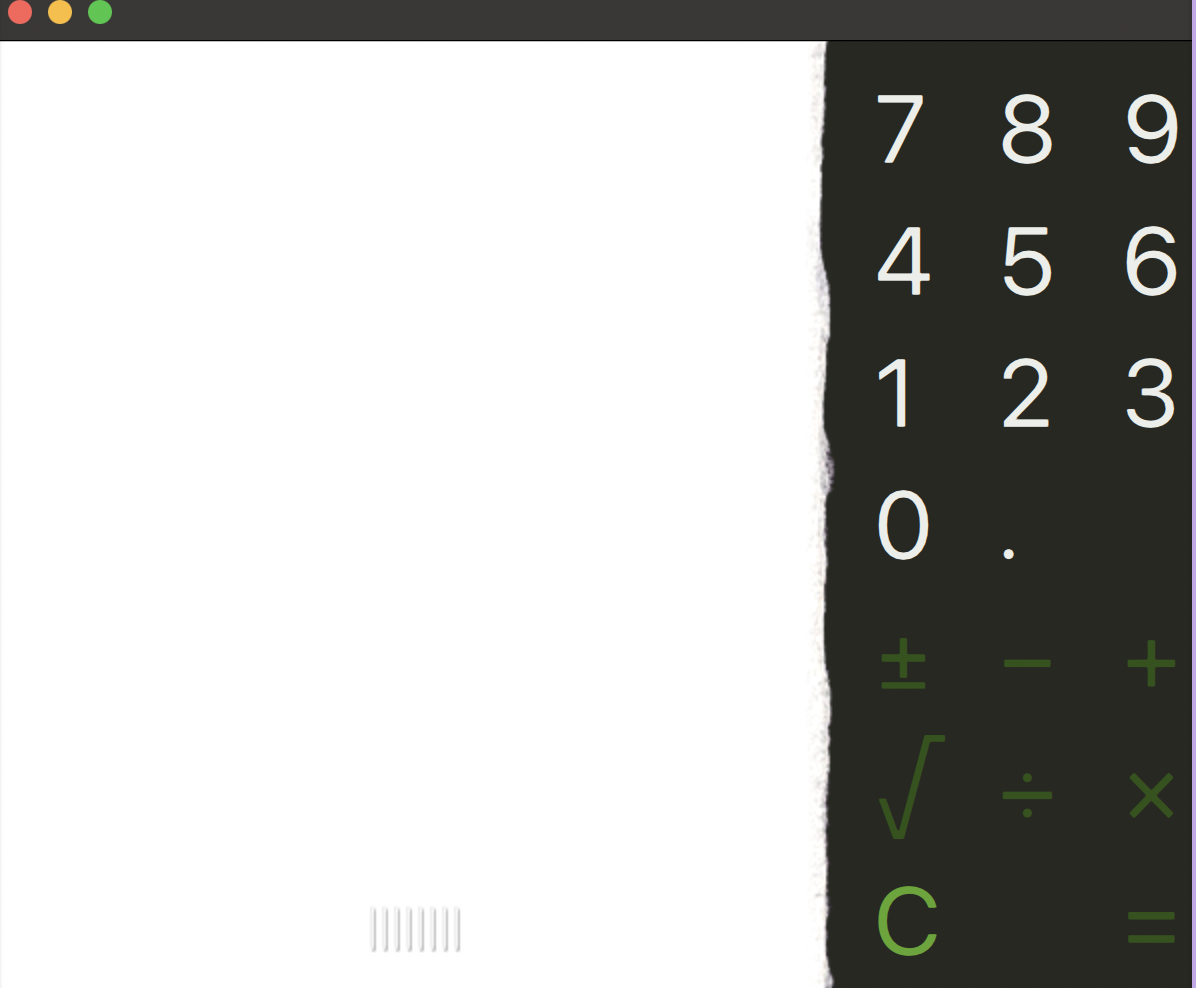
**Теоретичні відомості.**

Палітра компонент **VCL** – бібліотеки візуальних компонент **Borland C++ Builder** має ряд закладок, на яких згруповані піктограми всіх доступних стандартних компонент. Крім цього, є можливість створити власну компоненту і розмістити її на закладці, імпортувати набір компонент сторонніх розробників, змінювати розташування закладок. Розглянемо базові компоненти **Borland C++ Builder 6**.

Завдання для лабораторної роботи

1. Ознайомитись із палітрою компонент **Borland C++ Builder 6**.
2. Створити віконний проект, додати розглянуті візуальні компоненти.
3. Реалізувати калькулятор.

1.Ознайомився з палітрою компонент QT.



2. Створив віконний проект

// main.cpp

#include <QGuiApplication>

#include <QQmlEngine>

#include <QQmlFileSelector>

#include <QQuickView>

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

QCoreApplication::setOrganizationName("QtExamples");

QGuiApplication app(argc, argv);

QQuickView view;

view.connect(view.engine(), &QQmlEngine::quit, &app, &QCoreApplication::quit);

view.setSource(QUrl("qrc:/calculator.qml"));

if (view.status() == QQuickView::Error) return -1;

view.setResizeMode(QQuickView::SizeRootObjectToView);

view.show();

return app.exec();

}

// calculator.qml

import QtQuick

import "content"

import "content/calculator.js" as CalcEngine

Rectangle {

id: window

width: 320

height: 480

focus: true

color: "#272822"

onWidthChanged: controller.reload()

onHeightChanged: controller.reload()

function operatorPressed(operator) {

CalcEngine.operatorPressed(operator);

}

function digitPressed(digit) {

CalcEngine.digitPressed(digit);

}

function isButtonDisabled(op) {

return CalcEngine.disabled(op);

}

Item {

id: pad

width: 180

NumberPad {

id: numPad

y: 10

anchors.horizontalCenter: parent.horizontalCenter

}

}

AnimationController {

id: controller

animation: ParallelAnimation {

id: anim

NumberAnimation {

target: display

property: "x"

duration: 400

from: -16

to: window.width - display.width

easing.type: Easing.InOutQuad

}

NumberAnimation {

target: pad

property: "x"

duration: 400

from: window.width - pad.width

to: 0

easing.type: Easing.InOutQuad

}

SequentialAnimation {

NumberAnimation {

target: pad

property: "scale"

duration: 200

from: 1

to: 0.97

easing.type: Easing.InOutQuad

}

NumberAnimation {

target: pad

property: "scale"

duration: 200

from: 0.97

to: 1

easing.type: Easing.InOutQuad

}

}

}

}

Keys.onPressed: function (event) {

switch (event.key) {

case Qt.Key\_0:

digitPressed("0");

break;

case Qt.Key\_1:

digitPressed("1");

break;

case Qt.Key\_2:

digitPressed("2");

break;

case Qt.Key\_3:

digitPressed("3");

break;

case Qt.Key\_4:

digitPressed("4");

break;

case Qt.Key\_5:

digitPressed("5");

break;

case Qt.Key\_6:

digitPressed("6");

break;

case Qt.Key\_7:

digitPressed("7");

break;

case Qt.Key\_8:

digitPressed("8");

break;

case Qt.Key\_9:

digitPressed("9");

break;

case Qt.Key\_Plus:

operatorPressed("+");

break;

case Qt.Key\_Minus:

operatorPressed("-");

break;

case Qt.Key\_Asterisk:

operatorPressed("×");

break;

case Qt.Key\_Slash:

operatorPressed("÷");

break;

case Qt.Key\_Enter:

case Qt.Key\_Return:

operatorPressed("=");

break;

case Qt.Key\_Comma:

case Qt.Key\_Period:

digitPressed(".");

break;

case Qt.Key\_Backspace:

operatorPressed("backspace");

break;

}

}

Display {

id: display

x: -16

width: window.width - pad.width

height: parent.height

MouseArea {

id: mouseInput

property real startX: 0

property real oldP: 0

property bool rewind: false

anchors {

bottom: parent.bottom

left: parent.left

right: parent.right

}

height: 50

onPositionChanged: {

const reverse = startX > window.width / 2;

const mx = mapToItem(window, mouseInput.mouseX, mouseInput.mouseY).x;

const p = Math.abs((mx - startX) / (window.width - display.width));

rewind = p < oldP ? !reverse : reverse;

controller.progress = reverse ? 1 - p : p;

oldP = p;

}

onPressed: startX = mapToItem(window, mouseInput.mouseX, mouseInput.mouseY).x

onReleased: {

if (rewind)

controller.completeToBeginning();

else

controller.completeToEnd();

}

}

}

}

3.Реалізував калькулятор

Висновок: Виконуючи цю лабораторну роботу я навчився реалізовувати калькулятор, дослідив нові можливості QT.

**Лабораторна №3**

**Тема.**Огляд невізуальних компонент **Borland C++ Builder 6**. Створення проекту із невізуальними компонентами та їх використання.

**Мета.** Створити віконний проект та продемонструвати використання невізуальних компонент **C++ Builder**.

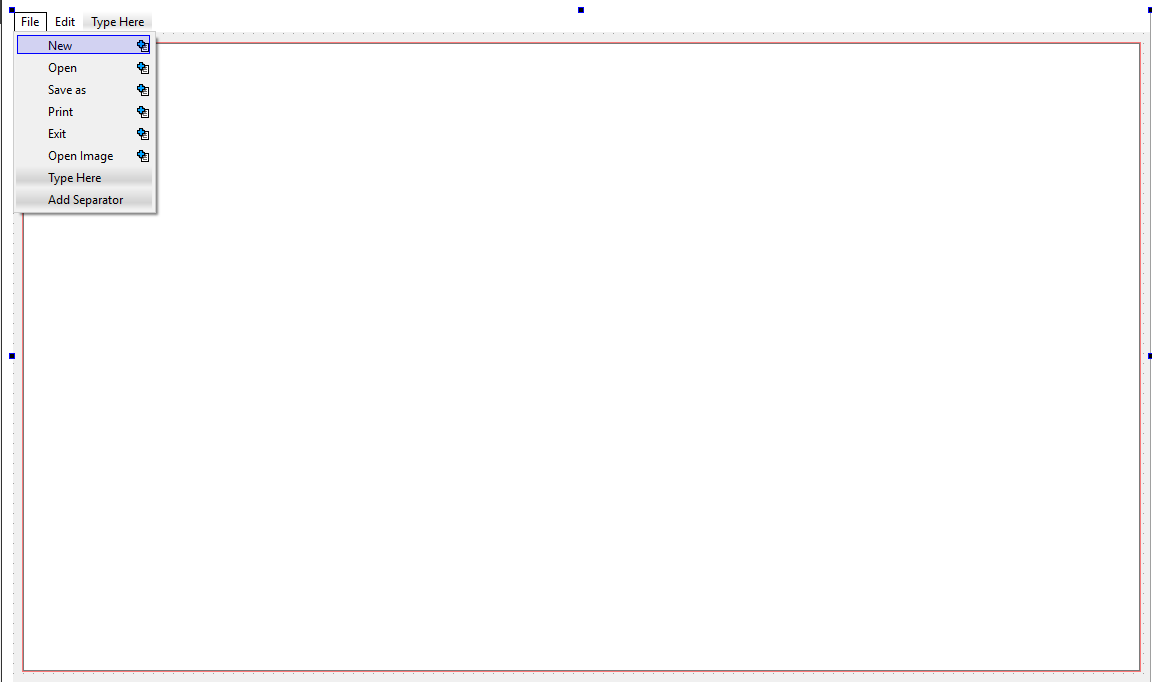
**Теоретичні відомості.**

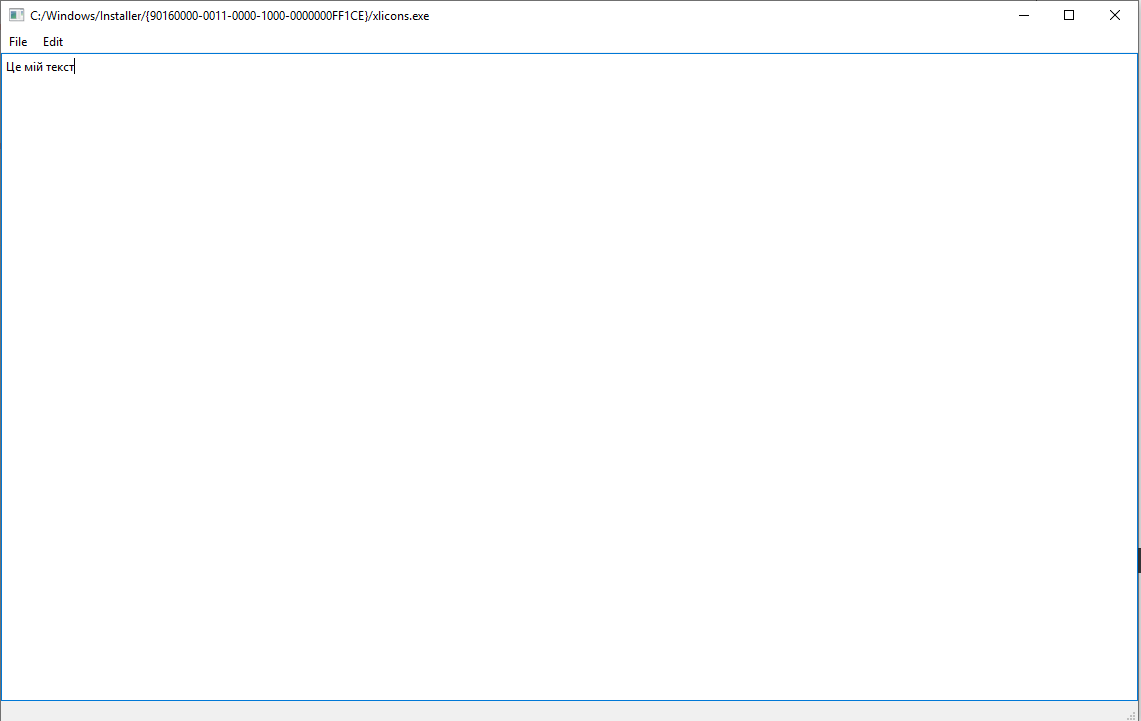
Невізуальні компоненти – це компоненти які є невидимі при виконанні застосування, а у режиміконструювання зображаються іконкою. Такі компоненти можна розміщати в будь-якому місці форми. Розглянемо роботу з базовими невізуальними компонентами **C++ Builder**.

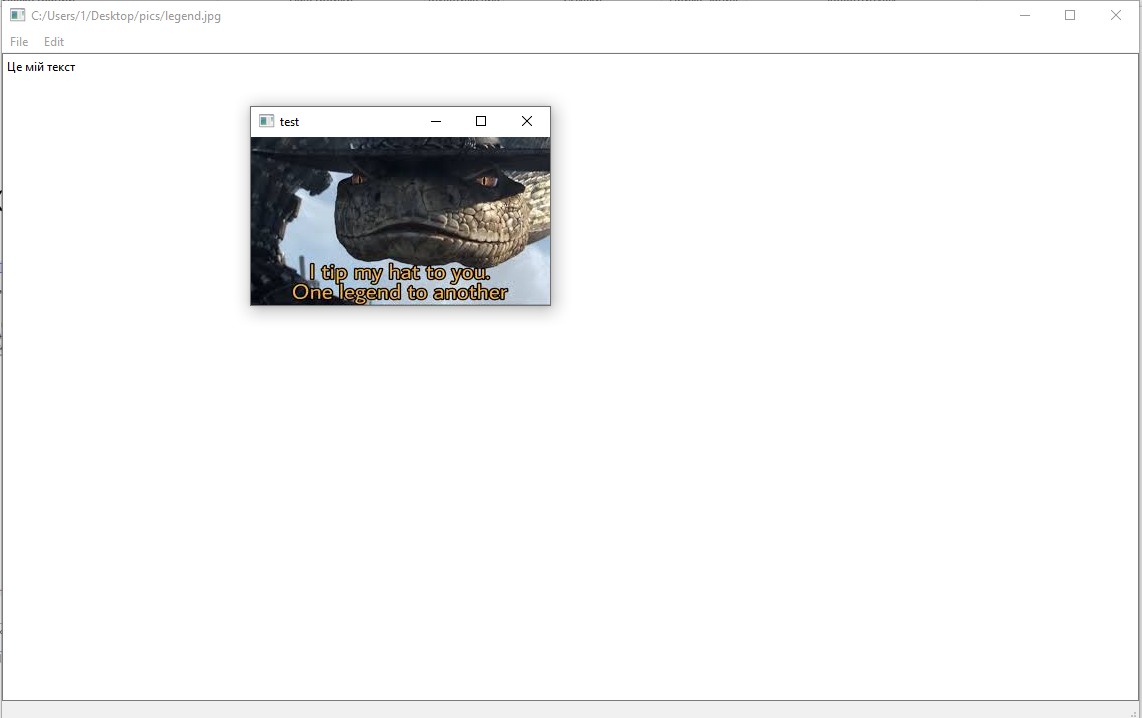
##### Завдання для лабораторної роботи

1. Створити віконний проект. Додати головне та контекстне меню, необхідні системні діалоги.
2. Реалізувати текстовий редактор і переглядач графічних файлів

1. Створив віконний проект, додав до нього головне та контекстне меню.







2.Реалізував текстовий редактор і переглядач графічних файлів.

//mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtCore>

#include <QtGui>

#include <QFile>

#include <QFileDialog>

#include <QTextStream>

#include <QMessageBox>

#include <QObject>

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

void AddRoot(QString name, QString Description);

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_actionNew\_triggered();

void on\_actionOpen\_triggered();

void on\_actionSave\_as\_triggered();

void on\_actionExit\_triggered();

void on\_actionCopy\_triggered();

void on\_actionPaste\_triggered();

void on\_actionCut\_triggered();

void on\_actionUndo\_triggered();

void on\_actionRedo\_triggered();

void on\_actionOpen\_Image\_triggered();

private:

Ui::MainWindow \*ui;

QString currentFile="";

};

#endif // MAINWINDOW\_H

//main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.show();

return a.exec();

}

//mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QtCore>

#include <QtGui>

#include <QLabel>

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

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setCentralWidget(*ui->textEdit*);

}

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

{

delete ui;

}

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

{

currentFile.clear();

ui->textEdit->setText(QString());

}

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

{

QString filename = QFileDialog::getOpenFileName(*(*this*)*,"Open the file");

QFile file(filename);

currentFile=filename;

if(!file.*open*(QIODevice::ReadOnly | QFile::Text)){

QMessageBox::warning(this,"Warning","Cannot open file:"+file.errorString());

return;

}

setWindowTitle(filename);

QTextStream in(*&file*);

QString text=in.readAll();

ui->textEdit->setText(text);

file.*close*();

}

void MainWindow::**on\_actionSave\_as\_triggered**()

{

QString filename = QFileDialog::getSaveFileName(this,"Save as");

QFile file(filename);

if(!file.*open*(QFile::WriteOnly | QFile::Text)){

QMessageBox::warning(this,"Warning","Cannot save file:"+file.errorString());

return;

}

currentFile=filename;

setWindowTitle(filename);

QTextStream out(*&file*);

QString text= ui->textEdit->toPlainText();

out<<text;

file.*close*();

}

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

{

QApplication::quit();

}

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

{

ui->textEdit->copy();

}

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

{

ui->textEdit->paste();

}

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

{

ui->textEdit->cut();

}

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

{

ui->textEdit->undo();

}

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

{

ui->textEdit->redo();

}

void MainWindow::**on\_actionOpen\_Image\_triggered**()

{

QString filename = QFileDialog::getOpenFileName(*(*this*)*,"Open the file");

QFile file(filename);

currentFile=filename;

if(!file.*open*(QIODevice::ReadOnly | QFile::Text)){

QMessageBox::warning(this,"Warning","Cannot open file:"+file.errorString());

return;

}

setWindowTitle(filename);

QPixmap image(currentFile);

QLabel \*imageLabel = new QLabel();

imageLabel->setPixmap(image);

imageLabel->show();

}

Висновок: виконуючи 3 лабораторну роботу я дізнався як реалізовувати текстовий редактор, працювати з класом «Діалог» у QT та як відкривати png та jpeg файли.

**Лабораторна робота №4**

**Тема.**Компоненти **Borland C++ Builder 6** для представлення даних.

**Мета.**Створити віконний проект та продемонструвати використання компонент призначених для відображення та опрацювання даних.

**Теоретичні відомості.**

Середовище розробки **Borland C++ Builder** має набір компонент які призначені для відображення, редагування, розміщення та опрацювання даних. Найпростіший спосіб відобразити дані – представити їх у вигляді таблиці. Для цього у **Borland C++ Builder** **6** є компонента **StringGrid**.

Завдання до лабораторної роботи:

1. Ознайомитись із компонентою **StringGrid**.
2. Реалізувати гру.

1.Я ознайомився із компонентою StringGrid

2.Реалізував гру

//mainwindow.cpp

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#define PLAYING\_FIELD\_SIZE\_TILES 16

#define PLAYING\_FIELD\_TILE\_SIZE 32

GamePalette myPalette = {

Qt::lightGray,

Qt::green,

Qt::darkGray,

Qt::red,

};

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), ui(new Ui::MainWindow) {

ui->setupUi(this);

setUpPlayingField();

setUpGameController();

gameController->startGame();

}

MainWindow::~MainWindow() {

delete ui;

deleteBrushes();

}

void MainWindow::deleteBrush(QBrush \*\*brush) {

if (\*brush != nullptr) {

delete \*brush;

\*brush = nullptr;

}

}

void MainWindow::deleteBrushes() {

deleteBrush(&backgroundBrush);

deleteBrush(&wallBrush);

deleteBrush(&fruitBrush);

deleteBrush(&snakeBrush);

}

void MainWindow::onTileUpdate(int x, int y, TileType type) {

auto cell = cellAt(x, y);

switch (type) {

case SNAKE\_BODY:

cell->setBackground(\*snakeBrush);

break;

case FRUIT:

cell->setBackground(\*fruitBrush);

break;

case EMPTY:

cell->setBackground(\*backgroundBrush);

break;

case WALL:

cell->setBackground(\*wallBrush);

break;

}

}

void MainWindow::onGameStopped(int score) {

QMessageBox gameLost(this);

gameLost.setWindowTitle("Game Over");

gameLost.setText(tr("Your score is %1").arg(score));

gameLost.setStandardButtons(QMessageBox::Retry);

auto button = gameLost.exec();

gameController->resetGame();

if (button == QMessageBox::Retry) {

gameController->startGame();

}

}

QTableWidgetItem \*MainWindow::cellAt(int x, int y) {

QTableWidgetItem \*item = ui->tableWidget->item(y, x);

if (item == nullptr) {

item = new QTableWidgetItem();

item->setTextAlignment(Qt::AlignCenter);

ui->tableWidget->setItem(y, x, item);

}

return item;

}

void MainWindow::keyPressEvent(QKeyEvent \*keyEvent) {

switch (keyEvent->key()) {

case Qt::Key\_W:

case Qt::Key\_K:

case Qt::Key\_Up:

gameController->snakeUp();

break;

case Qt::Key\_S:

case Qt::Key\_J:

case Qt::Key\_Down:

gameController->snakeDown();

break;

case Qt::Key\_A:

case Qt::Key\_H:

case Qt::Key\_Left:

gameController->snakeLeft();

break;

case Qt::Key\_D:

case Qt::Key\_L:

case Qt::Key\_Right:

gameController->snakeRight();

break;

}

keyEvent->accept();

}

void MainWindow::setUpPlayingField() {

ui->tableWidget->setColumnCount(PLAYING\_FIELD\_SIZE\_TILES);

ui->tableWidget->setRowCount(PLAYING\_FIELD\_SIZE\_TILES);

ui->tableWidget->horizontalHeader()->setMinimumSectionSize(0);

for (int i = 0; i < PLAYING\_FIELD\_SIZE\_TILES; ++i) {

ui->tableWidget->setColumnWidth(i, PLAYING\_FIELD\_TILE\_SIZE);

ui->tableWidget->setRowHeight(i, PLAYING\_FIELD\_TILE\_SIZE);

}

ui->tableWidget->horizontalHeader()->setVisible(false);

ui->tableWidget->verticalHeader()->setVisible(false);

QSize size = this->size();

#ifdef \_WIN32

size.setWidth(PLAYING\_FIELD\_SIZE\_TILES \* PLAYING\_FIELD\_TILE\_SIZE);

size.setHeight(PLAYING\_FIELD\_SIZE\_TILES \* PLAYING\_FIELD\_TILE\_SIZE +

ui->menubar->height());

#elif TARGET\_OS\_MAC

size.setWidth(PLAYING\_FIELD\_SIZE\_TILES \* PLAYING\_FIELD\_TILE\_SIZE + 2 \* 13);

size.setHeight(PLAYING\_FIELD\_SIZE\_TILES \* PLAYING\_FIELD\_TILE\_SIZE + 2 \* 13);

#endif

this->resize(size);

this->setMaximumSize(size);

this->setMinimumSize(size);

updatePalette(&myPalette);

}

void MainWindow::setUpGameController() {

// focus on window to receive key events

this->setFocusPolicy(Qt::StrongFocus);

this->setFocus();

gameController = new SnakeController(PLAYING\_FIELD\_SIZE\_TILES);

connect(gameController, SIGNAL(tileUpdateEvent(int, int, TileType)), this,

SLOT(onTileUpdate(int, int, TileType)));

connect(gameController, SIGNAL(gameStopEvent(int)), this,

SLOT(onGameStopped(int)));

gameController->resetGame();

gameController->setGameSpeed(3);

}

QBrush \*MainWindow::createBrush(Qt::GlobalColor color) {

return new QBrush(color);

}

void MainWindow::updatePalette(GamePalette \*newPalette) {

gamePalette = newPalette;

deleteBrushes();

backgroundBrush = createBrush(gamePalette->backgroundColor);

wallBrush = createBrush(gamePalette->wallColor);

snakeBrush = createBrush(gamePalette->snakeColor);

fruitBrush = createBrush(gamePalette->fruitColor);

if (gameController != nullptr) {

gameController->forceFieldRedraw();

}

}

//snake\_controller.cpp

#include "snake\_controller.h"

SnakeController::SnakeController(int fieldSize) {

this->fieldSize = fieldSize;

allocateField();

}

SnakeController::~SnakeController() { deleteField(); }

void SnakeController::forceFieldRedraw() {

for (int x = 0; x < fieldSize; ++x) {

for (int y = 0; y < fieldSize; ++y) {

emit tileUpdateEvent(x, y, tileToEnum(field[x][y]));

}

}

}

inline void SnakeController::updateTileAt(int x, int y, tile newTile) {

field[x][y] = newTile;

emit tileUpdateEvent(x, y, tileToEnum(newTile));

}

TileType SnakeController::tileToEnum(tile t) {

switch (t & TILE\_TYPE\_MASK) {

case EMPTY\_TILE:

return EMPTY;

case SNAKE\_TILE:

return SNAKE\_BODY;

case FRUIT\_TILE:

return FRUIT;

case WALL\_TILE:

return WALL;

default:

return EMPTY;

}

}

void SnakeController::resetGame() { updateGameState(READY\_FOR\_START); }

void SnakeController::startGame() {

if (gameState() != READY\_FOR\_START) return;

updateGameState(RUNNING);

}

void SnakeController::tick() {

tiletype type = getTileTypeInFront();

if (type == WALL\_TILE || type == SNAKE\_TILE) {

updateGameState(STOPPED);

return;

}

if (type == FRUIT\_TILE) {

placeFruit();

++snakeLength;

} else {

moveTail();

}

moveHead(newSnakeHeadDirection);

snakeHeadDirection = newSnakeHeadDirection;

}

void SnakeController::getRandomCoordinates(int \*x, int \*y) {

\*x = rand() % fieldSize;

\*y = rand() % fieldSize;

}

void SnakeController::moveHead(const direction headDirection) {

int px = snakeHeadX, py = snakeHeadY;

moveCoordinates(&snakeHeadX, &snakeHeadY, headDirection);

updateTileAt(px, py, convertHeadToBodyTile(field[px][py]));

updateTileAt(snakeHeadX, snakeHeadY,

createSnakeHeadTile(oppositeTo(headDirection)));

}

void SnakeController::moveTail() {

int tailX = snakeHeadX, tailY = snakeHeadY;

tile tile = field[tailX][tailY];

int px = 0, py = 0;

while (!isSnakeTail(tile)) {

px = tailX;

py = tailY;

direction dir = getDirection(tile);

moveCoordinates(&tailX, &tailY, dir);

tile = field[tailX][tailY];

}

updateTileAt(tailX, tailY, EMPTY\_TILE);

updateTileAt(px, py, createSnakeTailTile());

}

void SnakeController::updateGameState(GameState newState) {

this->state = newState;

switch (newState) {

case STOPPED:

killTimer(timerId);

emit gameStopEvent(snakeLength);

break;

case RUNNING:

timerId = startTimer(BASE\_TICK\_TIME\_MS / gameSpeed);

emit gameStartEvent();

case READY\_FOR\_START:

prepareFieldForStart();

break;

}

}

void SnakeController::allocateField() {

this->field = new tile \*[fieldSize];

for (int i = 0; i < fieldSize; ++i) {

field[i] = new tile[fieldSize];

}

}

void SnakeController::prepareFieldForStart() {

for (int i = 0; i < fieldSize; ++i) {

for (int j = 0; j < fieldSize; ++j) {

updateTileAt(i, j, EMPTY\_TILE);

}

}

// UL corner

updateTileAt(1, 0, WALL\_TILE);

updateTileAt(0, 0, WALL\_TILE);

updateTileAt(0, 1, WALL\_TILE);

updateTileAt(1, 1, WALL\_TILE);

// UR corner

updateTileAt(fieldSize - 1, 0, WALL\_TILE);

// LL corner

updateTileAt(0, fieldSize - 1, WALL\_TILE);

updateTileAt(1, fieldSize - 2, WALL\_TILE);

// lines

for (int y = 4; y < fieldSize - 4; ++y) {

updateTileAt(3, y, WALL\_TILE);

updateTileAt(7, y + 2, WALL\_TILE);

updateTileAt(fieldSize - 4, y, WALL\_TILE);

updateTileAt(y, 1, WALL\_TILE);

}

prepareSnake();

placeFruit();

}

void SnakeController::prepareSnake() {

snakeLength = INITIAL\_SNAKE\_LENGTH;

snakeHeadY = snakeHeadX = fieldSize / 3;

updateTileAt(snakeHeadX, snakeHeadY, createSnakeHeadTile(DIRECTION\_LEFT));

newSnakeHeadDirection = snakeHeadDirection = DIRECTION\_RIGHT;

for (int dx = 1; dx < INITIAL\_SNAKE\_LENGTH - 1; ++dx) {

const int bodyX = snakeHeadX - dx;

updateTileAt(bodyX, snakeHeadY, createSnakeBodyTile(DIRECTION\_LEFT));

}

const int tailX = snakeHeadX - INITIAL\_SNAKE\_LENGTH + 1;

updateTileAt(tailX, snakeHeadY, createSnakeTailTile());

}

void SnakeController::timerEvent(QTimerEvent \*event) {

if (event->timerId() != this->timerId) return;

tick();

}

void SnakeController::setGameSpeed(int speed) {

if (gameState() != RUNNING) {

this->gameSpeed = speed;

}

}

tiletype SnakeController::getTileTypeInFront() {

int x = snakeHeadX, y = snakeHeadY;

moveCoordinates(&x, &y, newSnakeHeadDirection);

if (x < 0 || x >= fieldSize) return WALL\_TILE;

if (y < 0 || y >= fieldSize) return WALL\_TILE;

return getTileType(field[x][y]);

}

void SnakeController::deleteField() {

for (int i = 0; i < fieldSize; ++i) {

delete[] field[i];

}

delete[] field;

}

void SnakeController::placeFruit() {

int x, y;

do {

getRandomCoordinates(&x, &y);

} while (field[x][y] != EMPTY\_TILE);

updateTileAt(x, y, createFruitTile());

}

void SnakeController::tryChangeDirection(direction newDirection) {

if (gameState() != RUNNING) return;

if (!isOppositeTo(snakeHeadDirection, newDirection)) {

newSnakeHeadDirection = newDirection;

}

}

//main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}





Висновки: виконуючи лабораторну роботу №4 я узагальнив свої знання та реалізував власну гру.