**Компьютерная академия «ШАГ»**

**Одесский филиал**

**Кафедра разработки программного обеспечения**

**КУРСОВОЙ ПРОЕКТ**

**“Easy type ”**

**Студента группы ЕКО-15-П1  
Пырхавко М. А.**

**Научный руководитель:**

**Соколова М. О.**

**(Одесса) 2017**

Аннотация

Программа представляет собой тренажёр для освоения десятипальцевого набора.

Пользователю доступны уровни по мере прохаждения которых он оттачивает свои навыки слепого набора. Когда точноть пользователя достигает 80% уровень считается завершнённым. Есть возможность выбрать язык изучения и свой текстовой файл для тренировки.

Содержание

Оглавление

[Техническое задание 5](#_Toc491897820)

[Использованные технологии 6](#_Toc491897821)

[Разработка структуры системы 7](#_Toc491897822)

[Разработка алгоритмов 8](#_Toc491897823)

[Разработка интерфейса 9](#_Toc491897824)

[Руководство пользователя 11](#_Toc491897825)

[Выводы 15](#_Toc491897826)

[Список использованной литературы 16](#_Toc491897827)

[Приложения 17](#_Toc491897828)

# 

Введение

Задачей курсовой работы было содание программного приложения . В качестве приложения мною была написан клавиатурный тренажёр “Easy type”.

Клавиату́рный тренажёр — вид [компьютерных программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) или онлайн-сервисов, предназначенных для обучения набору на [компьютерной клавиатуре](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BA%D0%BB%D0%B0%D0%B2%D0%B8%D0%B0%D1%82%D1%83%D1%80%D0%B0).

Целями тренажёра являются:

* научить [слепому методу печати](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B5%D0%BF%D0%BE%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4_%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D0%B8), в частности, задействовать для набора все десять [пальцев рук](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BB%D1%8C%D1%86%D1%8B_%D1%80%D1%83%D0%BA)
* увеличить скорость набора
* уменьшить количество опечаток
* улучшить ритмичность набора (что позволяет уменьшить усталость при наборе)

Процесс обучения построен в виде игры в которой пользователь выбирает уровень для прохождения которого нужно безошибочно и быстро набрать текст. В помощь пользователю показывается какую клавишу и каким плальцем нужно нажать. Не правильно введённый символ подсвечивается красным фоном.

# Техническое задание

1. Создать систему переключения языка интерфейса.
2. Разработать элемент управления показывающий клавишу для ввода и нужный для этого палец.
3. Разработаь элемент управления отображающий текст для ввода и пользовательский ввод.
4. Разработать элемент управления отображающий статистику.
5. Добавить возможность открыть свой текстовой файл.
6. Создать пользователей.
7. Добавить окно Авторизации пользовать.
8. Реализовать сохранения пройденных пользователем уровней в файл.

# Использованные технологии

В качестве технологий использовалось Windows API с использованием объектно-ориентированного подхода на базе языка С++. В качестве среды разработки использовалась Microsoft Visual Studio.

# Разработка структуры системы

Класс BaseDialogWindow

1. Класс от которого наследуются остальный классы.
2. Реализует перемаршутизацию сообщений.
3. Содержит методы для базовых операций.

Класс BaseTypingLanguage

1. Класс от которого наследуются производные языки ввода.
2. Необходим для добавления нового языка обучения.
3. Хранит информацию об уровнях.
4. Имеет методы для инициализации уровня.
   1. InitKeyBoard – инициализирует клавиатуру.
   2. LoadLevel – загружает текст уровня.

Класс LocalizationObject

1. По заросу возвращает локализированный текст.

Класс KeyboardDialog

1. Отображает клавишу для ввода.
2. Указывает пользователю на правильный палец необходимый для ввода.

Класс GameStatisticsDialog

1. Отображает Точность.
2. Отображает скорость ввода.

Класс TextDialog

1. Выводит текст.
2. Отображает курсор.
3. Подсвечивает не правильно введённые клавиши.

Класс MainDialog

1. Главный интерфейс пользователя
2. Отобаражает
   1. Меню.
   2. Поле ввода.
   3. Клавиатуру

Класс UserLogInDialog

1. Отвечает за авторизацию пользователя

# Разработка алгоритмов

В ходе создания программы были разработаны такие алгоритмы:

Подсчет скорости ввода:

Каждую секунду колличество введённых символов делится на количесво символов

Подсчет точности ввода:

Каждую секунду счиатется отношение правильного введённых символов к неправильно введнёмм символам.

Проверка правильности ввода:

Пользовательский ввод сверяется с текстом, и если пользователь допустил ошибку символ будет подсвечен красным цветом.

Определения клавиши для нажатия:

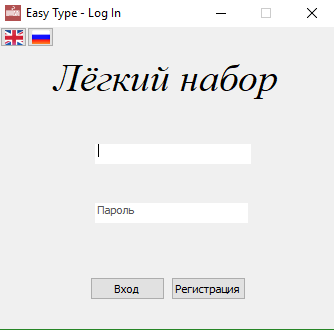
Каждому символу сотвествует клавиша на клавиатуре. По символу осщуествляется поиск необходимой клавиши. Если символ найден необходимая клавиша визуально на экране подсвечивается.

# Разработка интерфейса

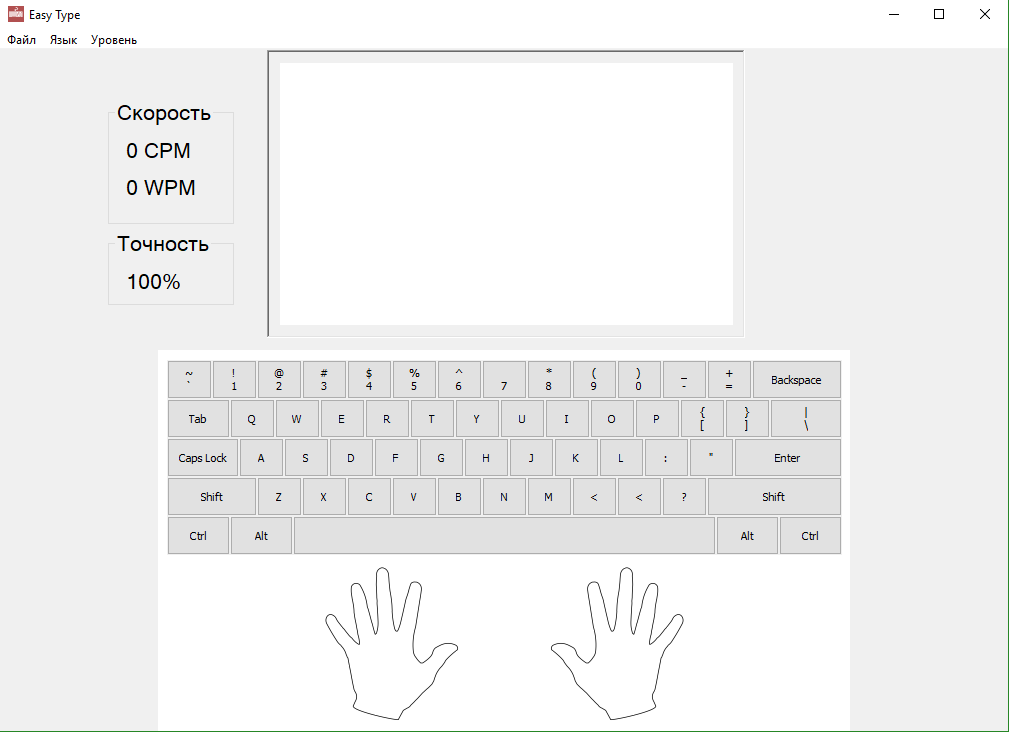
Приложение обладает простым и понятным интерфейсом.

Окно авторизации:

1. Флаги – смена языка интерфейса.
2. Вход – войти в приложение.
3. Регистрация – войти в приложение под новым пользователем.

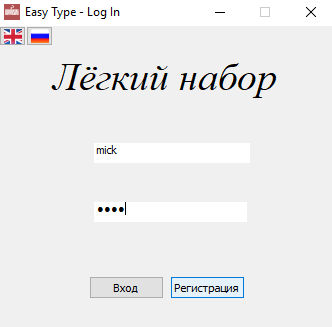


Главное окно:

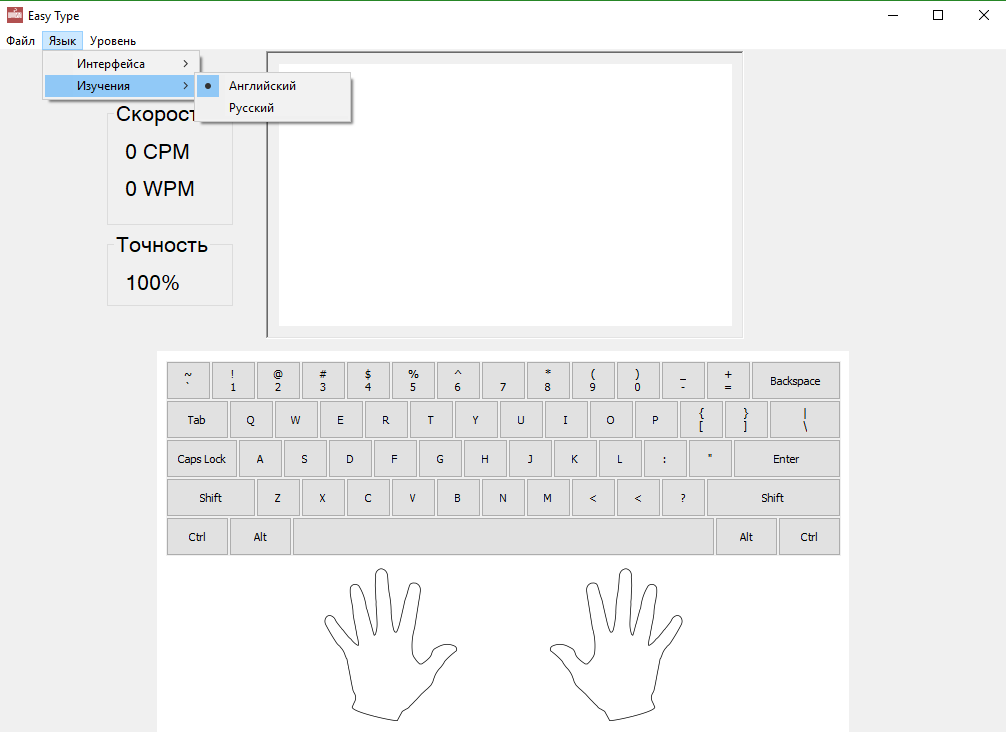


# Руководство пользователя

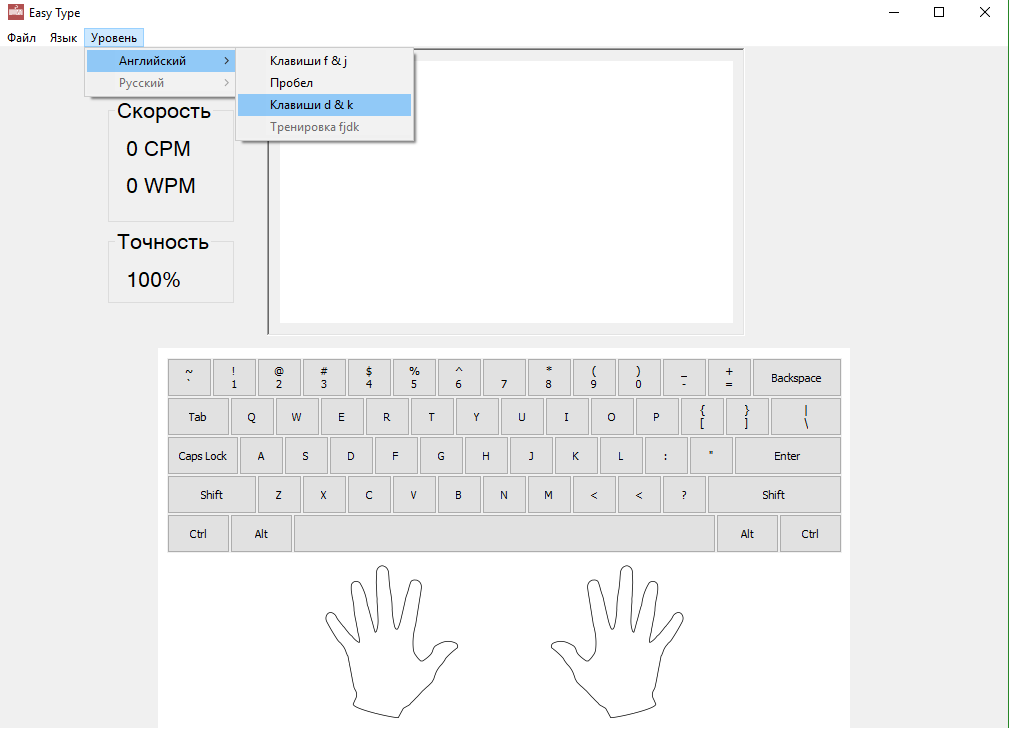
Регистрация нового пользователя



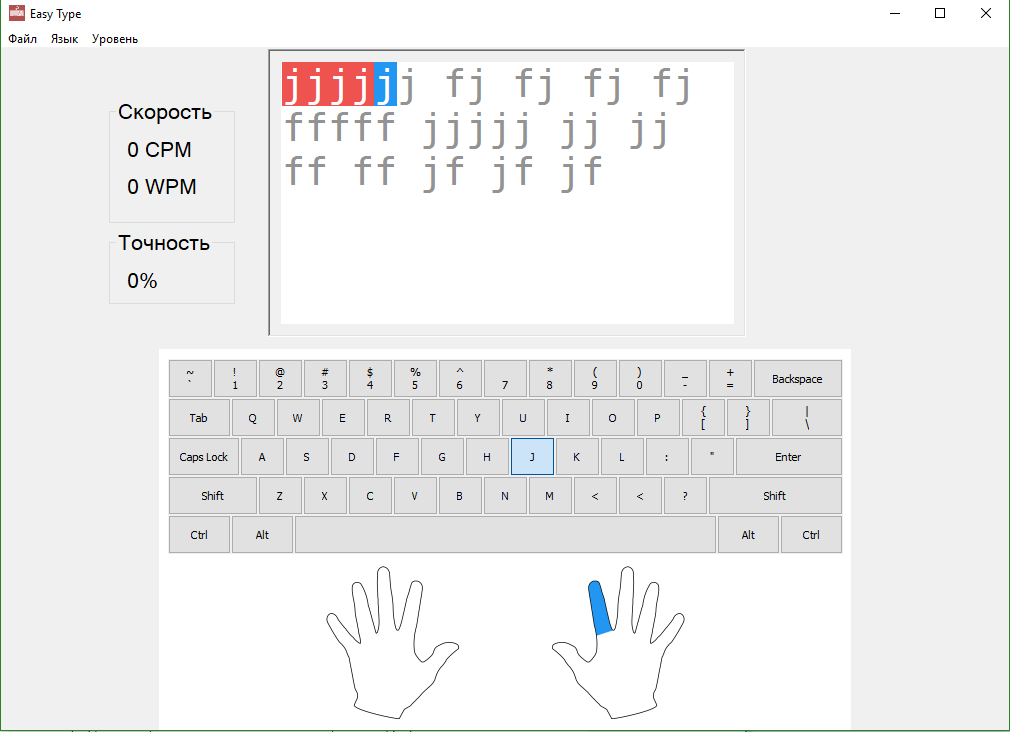
Выбор языка



Запуск уровня



Уровень



# Выводы

В ходе курсовой работы я имел возможность улучшить свои знания WIN32 API.

Я научился работать с Win32Api в объектно ориентированном стиле, и создавать свои элементы управления.

Основной сложностью было найти создание своих конторолов.

# Список использованной литературы

1. <https://msdn.microsoft.com/en-us/library/windows/desktop>
2. <http://www.functionx.com/win32/Lesson06.htm>
3. http://www.infernodevelopment.com/c-win32-api-simple-gui-wrapper#SimpleCWin32APIWrapperTutorial

# 

# Приложения

Приложение A: Исходный код

BaseDialogWindow.h

#pragma once

#include <windows.h>

#include <CommCtrl.h>

#include "DPI\_Scale.h"

#include "resource.h"

#include "LocalizationObject.h"

namespace CourseWork\_WinApi

{

class BaseDialogWindow

{

public:

virtual ~BaseDialogWindow();

BaseDialogWindow(INT nResource);

BOOL IsCreated();

HWND Create(HWND hwndParent = NULL, HINSTANCE hInstance = GetModuleHandle(NULL));

INT\_PTR CreateModal(HWND hwndParent = NULL, HINSTANCE hInstance = GetModuleHandle(NULL));

HWND GetWindowHandle();

INT GetResourceID();

HWND hWindow();

SIZE GetClientSize();

SIZE GetSize();

RECT GetRect();

POINT GetLocalPosition();

static tString GetText(HWND hWnd);

static POINT GetLocalPosition(HWND hWnd);

static SIZE GetClientSize(HWND hWnd);

static SIZE GetSize(HWND hWnd);

static INT GetMenuItemPosById(HMENU hMenu, INT id);

/\*~~~~~~~~~~~~~~ methods~~~~~~~~~~~~~~~~~~\*/

void Close();

protected:

HWND \_hwnd;

INT \_nResource;

INT \_nResult;

static DPI\_Scale \_dpiScale;

static LocalizationObject \_local;

/\*~~~~~~~~~~~~~~ methods~~~~~~~~~~~~~~~~~~\*/

BOOL isModal();

inline static BaseDialogWindow \* GetWindowObjectFromHandle(HWND hWnd);

virtual INT\_PTR DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) = 0;

private:

BOOL \_bModal;

//prohobit coping

BaseDialogWindow(const BaseDialogWindow &);

static INT\_PTR CALLBACK MessageRoute(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam); // routes messages from all windows

};

}

BaseDialogWindow.cpp

#include "stdafx.h"

#include "BaseDialogWindow.h"

namespace CourseWork\_WinApi

{

DPI\_Scale BaseDialogWindow::\_dpiScale;

LocalizationObject BaseDialogWindow::\_local;

BaseDialogWindow::~BaseDialogWindow()

{

Close();

\_hwnd = NULL;

}

BaseDialogWindow::BaseDialogWindow(INT nResource)

{

\_hwnd = NULL;

\_nResource = nResource;

}

BOOL BaseDialogWindow::IsCreated()

{

return ::IsWindow(\_hwnd);

}

HWND BaseDialogWindow::Create(HWND hwndParent, HINSTANCE hInstance)

{

if (IsCreated())

{

return \_hwnd;

}

\_bModal = false;

\_hwnd = CreateDialogParam(

hInstance,

MAKEINTRESOURCE(\_nResource),

hwndParent,

MessageRoute,

reinterpret\_cast<LPARAM>(this)

);

if (\_hwnd != NULL)

{

::ShowWindow(\_hwnd, SW\_SHOW);

}

return \_hwnd;

}

INT\_PTR BaseDialogWindow::CreateModal(HWND hwndParent, HINSTANCE hInstance)

{

\_bModal = true;

return DialogBoxParam(

hInstance,

MAKEINTRESOURCE(\_nResource),

hwndParent,

MessageRoute,

reinterpret\_cast<LONG>(this)

);

}

BOOL BaseDialogWindow::isModal()

{

return \_bModal;

}

inline BaseDialogWindow \* BaseDialogWindow::GetWindowObjectFromHandle(HWND hWnd)

{

return reinterpret\_cast<BaseDialogWindow \*>(GetWindowLong(hWnd, GWL\_USERDATA));

}

BaseDialogWindow::BaseDialogWindow(const BaseDialogWindow & window)

{

}

INT\_PTR CALLBACK BaseDialogWindow::MessageRoute(

HWND hwnd,

UINT uMsg,

WPARAM wParam,

LPARAM lParam)

{

BaseDialogWindow \*wnd = nullptr;

switch (uMsg)

{

case WM\_INITDIALOG:

{

SetWindowLong(hwnd, GWL\_USERDATA, lParam); // setting window user data as window pointer

reinterpret\_cast<BaseDialogWindow \*>(lParam)->\_hwnd = hwnd; // setting dialog box handle

HICON hIcon = LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(MAINICON));

SendMessage(hwnd, WM\_SETICON, ICON\_BIG, reinterpret\_cast<LPARAM>(hIcon));

SendMessage(hwnd, WM\_SETICON, ICON\_SMALL, reinterpret\_cast<LPARAM>(hIcon));

break;

}

default:

break;

}

wnd = GetWindowObjectFromHandle(hwnd);

if (wnd)

{

return wnd->DlgProc(hwnd, uMsg, wParam, lParam);

}

return false;

}

HWND BaseDialogWindow::GetWindowHandle()

{

return \_hwnd;

}

INT BaseDialogWindow::GetResourceID()

{

return \_nResource;

}

HWND BaseDialogWindow::hWindow()

{

return \_hwnd;

}

SIZE BaseDialogWindow::GetClientSize()

{

return GetClientSize(\_hwnd);

}

SIZE BaseDialogWindow::GetSize()

{

return GetSize(\_hwnd);

}

RECT BaseDialogWindow::GetRect()

{

RECT rect;

GetWindowRect(\_hwnd, &rect);

return rect;

}

POINT BaseDialogWindow::GetLocalPosition()

{

return GetLocalPosition(\_hwnd);

}

tString BaseDialogWindow::GetText(HWND hWindow)

{

size\_t length = GetWindowTextLength(hWindow) + 1;

LPTSTR str = new TCHAR[length];

GetWindowText(hWindow, str, length);

tString result(str);

delete str;

return result;

}

POINT BaseDialogWindow::GetLocalPosition(HWND hwnd)

{

RECT rect;

GetWindowRect(hwnd, &rect);

POINT point{ rect.left, rect.top };

HWND parentHwnd = reinterpret\_cast<HWND>(GetWindowLong(hwnd, GWL\_HWNDPARENT));

ScreenToClient(parentHwnd, &point);

return point;

}

SIZE BaseDialogWindow::GetClientSize(HWND hwnd)

{

RECT rect;

GetClientRect(hwnd, &rect);

return SIZE{ rect.right - rect.left, rect.bottom - rect.top };

}

SIZE BaseDialogWindow::GetSize(HWND hwnd)

{

RECT rect;

GetWindowRect(hwnd, &rect);

return SIZE{ rect.right - rect.left, rect.bottom - rect.top };

}

INT BaseDialogWindow::GetMenuItemPosById(HMENU hMenu, INT nId)

{

for (int i = GetMenuItemCount(hMenu); i >= 0; i--)

{

if (GetMenuItemID(hMenu, i) == nId)

return i;

}

return -1;

}

void BaseDialogWindow::Close()

{

if (IsCreated())

{

if (isModal())

EndDialog(\_hwnd, \_nResult);

else

DestroyWindow(\_hwnd);

}

}

}

GameStaticsDialog.h

#pragma once

#include "BaseDialogWindow.h"

namespace CourseWork\_WinApi

{

class GameStatisticsDialog :

public BaseDialogWindow

{

public:

GameStatisticsDialog();

~GameStatisticsDialog();

VOID UpdateCPM(INT cpm);

VOID UpdateAccuracy(INT accuracy);

VOID UpdateUILanguage();

INT CPM();

INT WPM();

INT Accuracy();

VOID Reset();

protected:

BOOL Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);

// Inherited via BaseDialogWindow

virtual INT\_PTR DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) override;

private:

HWND \_hwndCpm;

HWND \_hwndWpm;

HWND \_hwndAccuracy;

};

}

GameStatisticsDialog.cpp

#include "stdafx.h"

#include "GameStatisticsDialog.h"

#include <math.h>

namespace CourseWork\_WinApi

{

GameStatisticsDialog::GameStatisticsDialog()

:BaseDialogWindow(IDD\_DIALOG\_GAME\_STATICS)

{

}

GameStatisticsDialog::~GameStatisticsDialog()

{

}

VOID GameStatisticsDialog::UpdateCPM(INT nCPM)

{

tString cStr;

INT wpm;

cStr.append(to\_tstring(nCPM));

cStr.append(\_T(" CPM"));

SetWindowText(\_hwndCpm, cStr.c\_str());

cStr.clear();

if (nCPM)

{

wpm = nCPM / 5;

}

else

{

wpm = 0;

}

cStr.append(to\_tstring(wpm));

cStr.append(\_T(" WPM"));

SetWindowText(\_hwndWpm, cStr.c\_str());

}

VOID GameStatisticsDialog::UpdateAccuracy(INT nAccuracy)

{

tString str;

str.append(to\_tstring(nAccuracy));

str.append(\_T("%"));

SetWindowText(\_hwndAccuracy, str.c\_str());

}

VOID GameStatisticsDialog::UpdateUILanguage()

{

SetDlgItemText(

\_hwnd,

IDC\_STATIC\_SPEEDBOX,

\_local.QueryString(IDS\_STRING\_TYPING\_SPEED).c\_str()

);

SetDlgItemText(

\_hwnd,

IDC\_STATIC\_ACCURACYBOX,

\_local.QueryString(IDS\_STRING\_TYPING\_ACCURACY).c\_str()

);

}

INT GameStatisticsDialog::CPM()

{

return std::stoi(GetText(\_hwndCpm));

}

INT GameStatisticsDialog::WPM()

{

return std::stoi(GetText(\_hwndWpm));

}

INT GameStatisticsDialog::Accuracy()

{

return std::stoi(GetText(\_hwndAccuracy));

}

VOID GameStatisticsDialog::Reset()

{

UpdateAccuracy(100);

UpdateCPM(0);

}

BOOL GameStatisticsDialog::Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)

{

EnableWindow(hwnd, FALSE);

\_hwndCpm = GetDlgItem(hwnd, IDC\_STATIC\_WPM);

\_hwndWpm = GetDlgItem(hwnd, IDC\_STATIC\_CPM);

\_hwndAccuracy = GetDlgItem(hwnd, IDC\_STATIC\_ACCURACY);

Reset();

UpdateUILanguage();

return TRUE;

}

INT\_PTR GameStatisticsDialog::DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

HANDLE\_MSG(hwnd, WM\_INITDIALOG, Cls\_OnInitDialog);

}

return FALSE;

}

}

KeyboardDialog.h

#pragma once

#include "BaseDialogWindow.h"

namespace CourseWork\_WinApi

{

namespace Hand

{

enum Hand

{

NONE, RIGHT, LEFT

};

}

namespace Thinger

{

enum Thinger

{

NONE, THUMB, POINTER, MIDDLE, RING, LITTLE

};

}

class KeyboardDialog :

public BaseDialogWindow

{

public:

KeyboardDialog();

~KeyboardDialog();

virtual INT\_PTR DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

VOID Select(UINT id, BOOL shift = false);

VOID Reset();

protected:

/\*\*/

BOOL Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);

HBRUSH Cls\_OnCtlColor(HWND hwnd, HDC hdc, HWND hwndChild, int type);

/\*\*/

Hand::Hand DetermineHand(UINT controlId);

Thinger::Thinger DetermineThinger(UINT controlId);

VOID SetHandImage(Hand::Hand hand, Thinger::Thinger thinger);

BOOL UnselectLast();

private:

inline HBITMAP SetStaticBitmap(HWND hwndStatic, HBITMAP hBitmap);

/\*\*/

std::map<LONG, HBITMAP> \_bitmapsMap;

Hand::Hand \_lastHand;

UINT \_uLastId;

BOOL \_bLastTimeShift;

/\*settings\*/

BOOL \_bHands;

};

}

KeyboardDialog.cpp

#include "stdafx.h"

#include "KeyboardDialog.h"

namespace CourseWork\_WinApi

{

KeyboardDialog::KeyboardDialog()

:BaseDialogWindow(IDD\_DIALOG\_KEYBOARD)

{

\_lastHand = Hand::NONE;

\_bHands = TRUE;

\_uLastId = 0;

\_bLastTimeShift = FALSE;

HINSTANCE hInst = GetModuleHandle(NULL);

\_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_LEFT\_HAND)

);

\_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_THUMB] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_LEFT\_HAND\_THUMB)

);

\_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_MIDDLE] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_LEFT\_HAND\_MIDDLE)

);

\_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_RING] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_LEFT\_HAND\_RING)

);

\_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_LITTLE] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_LEFT\_HAND\_LITTLE)

);

\_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_POINTER] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_LEFT\_HAND\_POINTER)

);

\_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_RIGHT\_HAND)

);

\_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_THUMB] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_RIGHT\_HAND\_THUMB)

);

\_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_MIDDLE] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_RIGHT\_HAND\_MIDDLE)

);

\_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_RING] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_RIGHT\_HAND\_RING)

);

\_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_LITTLE] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_RIGHT\_HAND\_LITTLE)

);

\_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_POINTER] = LoadBitmap(

hInst,

MAKEINTRESOURCE(IDB\_BITMAP\_RIGHT\_HAND\_POINTER)

);

}

KeyboardDialog::~KeyboardDialog()

{

for (auto it = \_bitmapsMap.begin(); it != \_bitmapsMap.end(); ++it)

{

DeleteObject(it->second);

}

}

INT\_PTR KeyboardDialog::DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

HANDLE\_MSG(hwnd, WM\_INITDIALOG, Cls\_OnInitDialog);

HANDLE\_MSG(hwnd, WM\_CTLCOLORDLG, Cls\_OnCtlColor);

HANDLE\_MSG(hwnd, WM\_CTLCOLORSTATIC, Cls\_OnCtlColor);

}

return FALSE;

}

VOID KeyboardDialog::Select(UINT id, BOOL shift)

{

if (!\_bHands)

{

\_uLastId = 0;

return;

}

UnselectLast();

Hand::Hand hand = DetermineHand(id);

Thinger::Thinger thinger;

if (hand == Hand::NONE)

{

return;

}

Hand::Hand freeHand = (hand == Hand::LEFT ? Hand::RIGHT : Hand::LEFT);

// Highlighting the key

::SendDlgItemMessage(\_hwnd, id, BM\_SETSTATE, TRUE, NULL);

thinger = DetermineThinger(id);

SetHandImage(hand, thinger);

if (shift == TRUE)

{

::SendDlgItemMessage(

\_hwnd,

hand == Hand::LEFT ? IDC\_BUTTON\_SHIFT\_RIGHT : IDC\_BUTTON\_SHIFT\_LEFT,

BM\_SETSTATE,

TRUE,

NULL

);

SetHandImage(freeHand, Thinger::LITTLE);

}

else

{

SetHandImage(freeHand, Thinger::NONE);

}

\_uLastId = id;

\_lastHand = hand;

\_bLastTimeShift = shift;

}

VOID KeyboardDialog::Reset()

{

SetHandImage(Hand::LEFT, Thinger::NONE);

SetHandImage(Hand::RIGHT, Thinger::NONE);

UnselectLast();

}

BOOL KeyboardDialog::Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)

{

//SetActiveWindow(GetDlgItem(hwnd, IDC\_BUTTON\_KEY\_A));

::EnableWindow(hwnd, FALSE);

//SendDlgItemMessage(hwnd, IDC\_BUTTON\_KEY\_A, WM\_SETFOCUS, NULL, NULL);

SetHandImage(Hand::RIGHT, Thinger::NONE);

SetHandImage(Hand::LEFT, Thinger::NONE);

//ShowWindow(h, SW\_HIDE);

return TRUE;

}

HBRUSH KeyboardDialog::Cls\_OnCtlColor(HWND hwnd, HDC hdc, HWND hwndChild, int type)

{

return GetStockBrush(WHITE\_BRUSH);

}

Hand::Hand KeyboardDialog::DetermineHand(UINT controlId)

{

if (controlId < IDC\_BUTTON\_BACK\_QUOTE ||

controlId > IDC\_BUTTON\_CONTROL\_RIGHT)

{

return Hand::NONE;

}

if ((controlId >= IDC\_BUTTON\_BACK\_QUOTE && controlId <= IDC\_BUTTON\_DIGIT\_5) ||

(controlId >= IDC\_BUTTON\_TAB && controlId <= IDC\_BUTTON\_KEY\_T) ||

(controlId >= IDC\_BUTTON\_CAPSLOCK && controlId <= IDC\_BUTTON\_KEY\_G) ||

(controlId >= IDC\_BUTTON\_SHIFT\_LEFT && controlId <= IDC\_BUTTON\_KEY\_B) ||

(controlId >= IDC\_BUTTON\_CONTROL\_LEFT && controlId <= IDC\_BUTTON\_ALT\_LEFT))

{

return Hand::LEFT;

}

else if (controlId == IDC\_BUTTON\_SPACE)

{

if (\_lastHand == Hand::LEFT)

{

return Hand::RIGHT;

}

return Hand::LEFT;

}

return Hand::RIGHT;

}

Thinger::Thinger KeyboardDialog::DetermineThinger(UINT controlId)

{

Thinger::Thinger thinger;

switch (controlId)

{

/\*left\*/

case IDC\_BUTTON\_BACK\_QUOTE:

case IDC\_BUTTON\_TAB:

case IDC\_BUTTON\_CAPSLOCK:

case IDC\_BUTTON\_SHIFT\_LEFT:

case IDC\_BUTTON\_CONTROL\_LEFT:

case IDC\_BUTTON\_DIGIT\_1:

case IDC\_BUTTON\_KEY\_Q:

case IDC\_BUTTON\_KEY\_A:

case IDC\_BUTTON\_KEY\_Z:

thinger = Thinger::LITTLE;

break;

/\*left\*/

case IDC\_BUTTON\_DIGIT\_2:

case IDC\_BUTTON\_KEY\_W:

case IDC\_BUTTON\_KEY\_S:

case IDC\_BUTTON\_KEY\_X:

thinger = Thinger::RING;

break;

/\*left\*/

case IDC\_BUTTON\_DIGIT\_3:

case IDC\_BUTTON\_KEY\_E:

case IDC\_BUTTON\_KEY\_D:

case IDC\_BUTTON\_KEY\_C:

thinger = Thinger::MIDDLE;

break;

/\*left\*/

case IDC\_BUTTON\_DIGIT\_4:

case IDC\_BUTTON\_KEY\_R:

case IDC\_BUTTON\_KEY\_F:

case IDC\_BUTTON\_KEY\_V:

case IDC\_BUTTON\_DIGIT\_5:

case IDC\_BUTTON\_KEY\_T:

case IDC\_BUTTON\_KEY\_G:

case IDC\_BUTTON\_KEY\_B:

thinger = Thinger::POINTER;

break;

/\*middle\*/

case IDC\_BUTTON\_SPACE:

case IDC\_BUTTON\_ALT\_LEFT:

case IDC\_BUTTON\_ALT\_RIGHT:

thinger = Thinger::THUMB;

break;

/\*right\*/

case IDC\_BUTTON\_DIGIT\_6:

case IDC\_BUTTON\_KEY\_Y:

case IDC\_BUTTON\_KEY\_H:

case IDC\_BUTTON\_KEY\_N:

case IDC\_BUTTON\_DIGIT\_7:

case IDC\_BUTTON\_KEY\_U:

case IDC\_BUTTON\_KEY\_J:

case IDC\_BUTTON\_KEY\_M:

thinger = Thinger::POINTER;

break;

/\*right\*/

case IDC\_BUTTON\_DIGIT\_8:

case IDC\_BUTTON\_KEY\_I:

case IDC\_BUTTON\_KEY\_K:

case IDC\_BUTTON\_COMMA:

thinger = Thinger::MIDDLE;

break;

/\*right\*/

case IDC\_BUTTON\_DIGIT\_9:

case IDC\_BUTTON\_KEY\_O:

case IDC\_BUTTON\_KEY\_L:

case IDC\_BUTTON\_PERIOD:

thinger = Thinger::RING;

break;

/\*right\*/

case IDC\_BUTTON\_DIGIT\_0:

case IDC\_BUTTON\_KEY\_P:

case IDC\_BUTTON\_SEMICOLON:

case IDC\_BUTTON\_SLASH:

case IDC\_BUTTON\_MINUS:

case IDC\_BUTTON\_BRACKET\_LEFT:

case IDC\_BUTTON\_QUOTE:

case IDC\_BUTTON\_EQUAL:

case IDC\_BUTTON\_BRACKET\_RIGHT:

case IDC\_BUTTON\_BACKSPACE:

case IDC\_BUTTON\_BACKSLASH:

case IDC\_BUTTON\_ENTER:

case IDC\_BUTTON\_SHIFT\_RIGHT:

case IDC\_BUTTON\_CONTROL\_RIGHT:

thinger = Thinger::LITTLE;

break;

default:

thinger = Thinger::NONE;

break;

}

return thinger;

}

BOOL KeyboardDialog::UnselectLast()

{

if (\_uLastId != 0)

{

::SendDlgItemMessage(\_hwnd, \_uLastId, BM\_SETSTATE, FALSE, NULL);

if (\_bLastTimeShift)

{

::SendDlgItemMessage(

\_hwnd,

\_lastHand == Hand::LEFT ? IDC\_BUTTON\_SHIFT\_RIGHT : IDC\_BUTTON\_SHIFT\_LEFT,

BM\_SETSTATE,

FALSE,

NULL

);

}

return true;

}

return false;

}

VOID KeyboardDialog::SetHandImage(Hand::Hand hand, Thinger::Thinger thinger)

{

if (hand == Hand::NONE)

{

return;

}

HWND hStatic;

if (hand == Hand::LEFT)

{

hStatic = GetDlgItem(\_hwnd, IDC\_STATIC\_LEFT\_HAND);;

switch (thinger)

{

case Thinger::THUMB:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_THUMB]);

break;

case Thinger::POINTER:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_POINTER]);

break;

case Thinger::MIDDLE:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_MIDDLE]);

break;

case Thinger::RING:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_RING]);

break;

case Thinger::LITTLE:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND\_LITTLE]);

break;

default:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_LEFT\_HAND]);

break;

}

}

else

{

hStatic = GetDlgItem(\_hwnd, IDC\_STATIC\_RIGHT\_HAND);;

switch (thinger)

{

case Thinger::THUMB:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_THUMB]);

break;

case Thinger::POINTER:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_POINTER]);

break;

case Thinger::MIDDLE:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_MIDDLE]);

break;

case Thinger::RING:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_RING]);

break;

case Thinger::LITTLE:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND\_LITTLE]);

break;

default:

SetStaticBitmap(hStatic, \_bitmapsMap[IDB\_BITMAP\_RIGHT\_HAND]);

break;

}

}

}

HBITMAP KeyboardDialog::SetStaticBitmap(HWND hwndStatic, HBITMAP hBitmap)

{

LRESULT lResult = SendMessage(

hwndStatic,

STM\_SETIMAGE,

IMAGE\_BITMAP,

reinterpret\_cast<LPARAM>(hBitmap)

);

return reinterpret\_cast<HBITMAP>(lResult);

}

}

MainDialog.h

#pragma once

#include "stdafx.h"

#include "BaseDialogWindow.h"

#include "UserManagement.h"

#include "KeyboardDialog.h"

#include "TextDialog.h"

#include "BaseTypingLanguage.h"

#include "GameStatisticsDialog.h"

namespace CourseWork\_WinApi

{

class MainDialog :

public BaseDialogWindow

{

public:

MainDialog();

~MainDialog();

INT\_PTR DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

protected:

BOOL Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);

VOID Cls\_OnMove(HWND hwnd, INT x, INT y);

VOID Cls\_OnSize(HWND hwnd, UINT state, INT cx, INT cy);

VOID Cls\_OnGetMinMaxInfo(HWND hwnd, LPMINMAXINFO lpMinMaxInfo);

VOID Cls\_OnKey(HWND hwnd, UINT vk, BOOL fDown, INT cRepeat, UINT flags);

VOID Cls\_OnChar(HWND hwnd, TCHAR ch, INT cRepeat);

HBRUSH Cls\_OnCtlColor(HWND hwnd, HDC hdc, HWND hwndChild, INT type);

VOID Cls\_OnCommand(HWND hwnd, INT id, HWND hwndCtl, UINT codeNotify);

VOID Cls\_OnDestroy(HWND hwnd);

VOID Cls\_OnClose(HWND hwnd);

VOID Cls\_OnInitMenuPopup(HWND hwnd, HMENU hMenu, UINT item, BOOL fSystemMenu);

VOID Cls\_OnMenuSelect(HWND hwnd, HMENU hmenu, INT item, HMENU hmenuPopup, UINT flags);

VOID Cls\_OnTimer(HWND hwnd, UINT id);

/\*\*/

VOID OnOpenFile();

VOID UpdateMenu();

VOID AlignElements();

private:

/\*\*/

VOID UpdateSpeedStat();

BOOL CheckIfLevel(INT id);

VOID OnLevelEnd();

/\*\*/

UserManagement \_userManagement;

BaseTypingLanguage \* \_typingLanguage;

KeyboardDialog \_keyboard;

TextDialog \_text;

GameStatisticsDialog \_statistics;

INT \_iCurrentLevel;

DWORD \_millisecondsStart;

/\*\*/

HMENU \_hmKeyboardLanguage;

HMENU \_hmUILanguage;

HMENU \_hmLevel;

HMENU \_hmRuLevel;

HMENU \_hmEnLevel;

};

}

MainDialog.cpp

#include "stdafx.h"

#include "MainDialog.h"

#include "UserLogInDialog.h"

#include <Richedit.h>

#include "RussianTypingLanguage.h"

#include "EnglishTypingLanguage.h"

#include <fstream>

#include <algorithm>

#include <codecvt>

#include <string>

#define IDT\_TIMER\_SPEED USER\_TIMER\_MINIMUM + 1

namespace CourseWork\_WinApi

{

MainDialog::MainDialog()

:BaseDialogWindow(IDD\_DIALOG\_MAIN)

{

\_typingLanguage = new EnglishTypingLanguage();

\_local.SetLanguage(Language::Russian);

\_iCurrentLevel = 0;

\_millisecondsStart = 0;

}

MainDialog::~MainDialog()

{

delete \_typingLanguage;

}

INT\_PTR MainDialog::DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

HANDLE\_MSG(hwnd, WM\_CLOSE, Cls\_OnClose);

HANDLE\_MSG(hwnd, WM\_INITDIALOG, Cls\_OnInitDialog);

HANDLE\_MSG(hwnd, WM\_DESTROY, Cls\_OnDestroy);

HANDLE\_MSG(hwnd, WM\_MOVE, Cls\_OnMove);

HANDLE\_MSG(hwnd, WM\_SIZE, Cls\_OnSize);

HANDLE\_MSG(hwnd, WM\_GETMINMAXINFO, Cls\_OnGetMinMaxInfo);

HANDLE\_MSG(hwnd, WM\_KEYDOWN, Cls\_OnKey);

HANDLE\_MSG(hwnd, WM\_CHAR, Cls\_OnChar);

HANDLE\_MSG(hwnd, WM\_CTLCOLORDLG, Cls\_OnCtlColor);

HANDLE\_MSG(hwnd, WM\_COMMAND, Cls\_OnCommand);

HANDLE\_MSG(hwnd, WM\_INITMENUPOPUP, Cls\_OnInitMenuPopup);

HANDLE\_MSG(hwnd, WM\_MENUSELECT, Cls\_OnMenuSelect);

HANDLE\_MSG(hwnd, WM\_TIMER, Cls\_OnTimer);

}

return FALSE;

}

VOID MainDialog::Cls\_OnDestroy(HWND hwnd)

{

PostQuitMessage(NULL);

}

VOID MainDialog::Cls\_OnClose(HWND hwnd)

{

INT result = MessageBox(

\_hwnd,

\_local.QueryString(IDS\_STRING\_PROMPT\_CLOSE).c\_str(),

\_local.QueryString(IDS\_STRING\_PROMPT\_CLOSE\_TITLE).c\_str(),

MB\_ICONINFORMATION | MB\_OKCANCEL

);

if (result == IDOK)

{

Close();

}

}

VOID MainDialog::Cls\_OnInitMenuPopup(HWND hwnd, HMENU hMenu, UINT item, BOOL fSystemMenu)

{

if (hMenu == \_hmKeyboardLanguage)

{

UINT uCheck = \_typingLanguage->Language() == Language::English

? ID\_LEARNING\_ENGLISH : ID\_LEARNING\_RUSSIAN;

CheckMenuRadioItem(

hMenu,

ID\_LEARNING\_ENGLISH,

ID\_LEARNING\_RUSSIAN,

uCheck,

MF\_BYCOMMAND

);

}

else if (hMenu == \_hmUILanguage)

{

UINT uCheck = \_local.GetLanguage() == Language::English

? ID\_INTERFACE\_ENGLISH : ID\_INTERFACE\_RUSSIAN;

CheckMenuRadioItem(

hMenu,

ID\_INTERFACE\_ENGLISH,

ID\_INTERFACE\_RUSSIAN,

uCheck,

MF\_BYCOMMAND

);

}

else if (hMenu == \_hmLevel)

{

if (\_typingLanguage->Language() == Language::English)

{

EnableMenuItem(hMenu, 0, MF\_ENABLED | MF\_BYPOSITION);

EnableMenuItem(hMenu, 1, MF\_DISABLED | MF\_BYPOSITION);

}

else

{

EnableMenuItem(hMenu, 1, MF\_ENABLED | MF\_BYPOSITION);

EnableMenuItem(hMenu, 0, MF\_DISABLED | MF\_BYPOSITION);

}

}

else if (hMenu == \_hmEnLevel)

{

INT level = \_userManagement.GetCurrentUser().GetLevel(Language::English);

INT itemsCount = GetMenuItemCount(hMenu);

for (int i = 1; i <= itemsCount; i++)

{

EnableMenuItem(

hMenu,

GetMenuItemID(hMenu, i - 1),

i > level ? MF\_GRAYED : MF\_ENABLED

);

}

}

else if (hMenu == \_hmRuLevel)

{

INT level = \_userManagement.GetCurrentUser().GetLevel(Language::Russian);

INT itemsCount = GetMenuItemCount(hMenu);

for (int i = 1; i <= itemsCount; i++)

{

EnableMenuItem(

hMenu,

GetMenuItemID(hMenu, i - 1),

i > level ? MF\_GRAYED : MF\_ENABLED

);

}

}

}

VOID MainDialog::Cls\_OnMenuSelect(HWND hwnd, HMENU hmenu, INT item, HMENU hmenuPopup, UINT flags)

{

}

VOID MainDialog::Cls\_OnTimer(HWND hwnd, UINT id)

{

switch (id)

{

case IDT\_TIMER\_SPEED:

UpdateSpeedStat();

break;

default:

break;

}

}

VOID MainDialog::UpdateSpeedStat()

{

DWORD milliseconds = GetTickCount();

DOUBLE minutes = ((milliseconds - \_millisecondsStart) / 1000.0) / 60.0;

if (\_text.CursorPos() > 0)

{

\_statistics.UpdateCPM(static\_cast<INT>(\_text.CursorPos() / minutes));

}

else

{

\_statistics.UpdateCPM(0);

}

}

BOOL MainDialog::Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)

{

UserLogInDialog \* userLoginDialog = new UserLogInDialog(\_userManagement);

BOOL bResult = (userLoginDialog->CreateModal() == IDOK);

delete userLoginDialog;

if (bResult != IDOK)

{

Close();

}

UpdateMenu();

\_keyboard.Create(\_hwnd);

\_text.Create(\_hwnd);

\_statistics.Create(\_hwnd);

\_typingLanguage->InitKeyBoard(\_keyboard);

// \u23ce\offset

SetWindowPos(\_keyboard.hWindow(), NULL, 30, 300, 0, 0, SWP\_NOSIZE);

SetWindowPos(hwnd, NULL, 0, 0, 1024, 740, SWP\_NOMOVE);

//\_text.SetText(TEXT("fffff \u23ce\offset jjjjjj fj fj fj fj fffff jjjjj jj jj ff ff jf jf jf"));

return TRUE;

}

VOID MainDialog::Cls\_OnMove(HWND hwnd, INT x, INT y)

{

return VOID();

}

VOID MainDialog::Cls\_OnSize(HWND hwnd, UINT state, INT cx, INT cy)

{

switch (state)

{

case SIZE\_RESTORED:

case SIZE\_MAXIMIZED:

break;

}

if (state == SIZE\_RESTORED || state == SIZE\_MAXIMIZED)

{

AlignElements();

}

}

void MainDialog::Cls\_OnGetMinMaxInfo(HWND hwnd, LPMINMAXINFO lpMinMaxInfo)

{

lpMinMaxInfo->ptMinTrackSize.x = 1024;

lpMinMaxInfo->ptMinTrackSize.y = 740;

}

VOID MainDialog::Cls\_OnKey(HWND hwnd, UINT vk, BOOL fDown, INT cRepeat, UINT flags)

{

}

VOID MainDialog::Cls\_OnChar(HWND hwnd, TCHAR ch, INT cRepeat)

{

if (!\_iCurrentLevel)

{

return;

}

if (\_millisecondsStart == 0)

{

\_millisecondsStart = GetTickCount();

SetTimer(\_hwnd, IDT\_TIMER\_SPEED, 1000, NULL);

}

const TypeButtonInfo \* nextCharInfo = (\*\_typingLanguage)[\_text.NextCharacter()];

if (nextCharInfo != NULL)

{

\_keyboard.Select(nextCharInfo->Id(), nextCharInfo->ShiftPressed());

}

else

{

\_keyboard.Reset();

}

if (ch == \_T('\b'))

{

\_text.Undo();

}

else if (ch == \_T('\r') && \_text.CurrentCharacter() == \_T('\u23ce'))

{

\_text.Right();

}

else if (ch == \_text.CurrentCharacter())

{

\_text.Right();

}

else

{

\_text.Wrong();

}

if (\_text.WrongSymbolsCount() > 0 && \_text.CursorPos() > 0)

{

\_statistics.UpdateAccuracy(

100 \* \_text.RightSymbolsCount() / \_text.CursorPos()

);

}

else

{

\_statistics.UpdateAccuracy(100);

}

if (\_text.isEnd())

{

OnLevelEnd();

}

}

HBRUSH MainDialog::Cls\_OnCtlColor(HWND hwnd, HDC hdc, HWND hwndChild, INT type)

{

return HBRUSH();

}

VOID MainDialog::Cls\_OnCommand(HWND hwnd, INT id, HWND hwndCtl, UINT codeNotify)

{

if (codeNotify == 0)

{

switch (id)

{

case ID\_INTERFACE\_ENGLISH:

\_local.SetLanguage(Language::English);

\_statistics.UpdateUILanguage();

UpdateMenu();

break;

case ID\_INTERFACE\_RUSSIAN:

\_local.SetLanguage(Language::Russian);

\_statistics.UpdateUILanguage();

UpdateMenu();

break;

case ID\_LEARNING\_ENGLISH:

if (\_typingLanguage->Language() == Language::English)

{

break;

}

delete \_typingLanguage;

\_typingLanguage = new EnglishTypingLanguage();

\_typingLanguage->InitKeyBoard(\_keyboard);

\_iCurrentLevel = 0;

\_text.Reset();

KillTimer(\_hwnd, IDT\_TIMER\_SPEED);

break;

case ID\_LEARNING\_RUSSIAN:

if (\_typingLanguage->Language() == Language::Russian)

{

break;

}

delete \_typingLanguage;

\_typingLanguage = new RussianTypingLanguage();

\_typingLanguage->InitKeyBoard(\_keyboard);

\_iCurrentLevel = 0;

\_text.Reset();

KillTimer(\_hwnd, IDT\_TIMER\_SPEED);

break;

case ID\_FILE\_OPEN:

OnOpenFile();

break;

default:

CheckIfLevel(id);

break;

}

}

return VOID();

}

VOID MainDialog::OnOpenFile()

{

OPENFILENAME OpenFileName = {};

LARGE\_INTEGER fileSize;

LPTSTR szFileFullPath = (LPTCH)LocalAlloc(LMEM\_FIXED, sizeof(TCHAR) \* MAX\_PATH);

ZeroMemory(szFileFullPath, sizeof(TCHAR) \* MAX\_PATH);

OpenFileName.lStructSize = sizeof(OPENFILENAME);

OpenFileName.hwndOwner = \_hwnd;

OpenFileName.lpstrFilter = TEXT("Text Files(\*.txt)\0\*.txt\0All Files(\*.\*)\0\*.\*\0");

OpenFileName.lpstrFile = szFileFullPath;

OpenFileName.nMaxFile = MAX\_PATH;

OpenFileName.lpstrInitialDir = TEXT("C:\\");

OpenFileName.Flags = OFN\_CREATEPROMPT | OFN\_PATHMUSTEXIST | OFN\_HIDEREADONLY;

if (GetOpenFileName(&OpenFileName))

{

std::wfstream fileStream;

std::locale locale(std::locale("Russian"), new std::codecvt\_utf8<TCHAR>());

fileStream.imbue(locale);

tString text;

fileStream.open(szFileFullPath, std::ios::in);

if (fileStream.is\_open())

{

/\*std::wfstream::streampos begin, end;

begin = fileStream.tellg();

fileStream.seekg(0, fileStream.end);

end = fileStream.tellg();

WORD size = (end - begin) + 1;

if (size == 0)

{

fileStream.close();

return;

}

fileStream.seekg(0, fileStream.beg);

LPTCH lptszTempBuff = (LPTCH)LocalAlloc(LMEM\_FIXED, sizeof(TCHAR) \* (size + 1));

fileStream.read(lptszTempBuff, size);

lptszTempBuff[size - 1] = TEXT('\0');

text = tString(lptszTempBuff);

LocalFree(lptszTempBuff);

fileStream.close();

\*/

tString str = \_T("");

while (std::getline(fileStream, str))

{

if (str[0] == 65279)

{

str.replace(0, 1, \_TEXT(""));

}

text += str;

text.push\_back(\_T('\n'));

}

tString search = \_T("\n");

tString replacement = \_T("\u23ce\n");

tString::size\_type offset = 0;

while ((offset = text.find(search, offset)) != std::string::npos)

{

text.replace(offset, search.size(), replacement);

offset += replacement.size();

}

\_iCurrentLevel = -1;

\_text.SetText(text);

\_keyboard.Reset();

const TypeButtonInfo \* nextCharInfo = (\*\_typingLanguage)[\_text.CurrentCharacter()];

if (nextCharInfo != NULL)

{

\_keyboard.Select(nextCharInfo->Id(), nextCharInfo->ShiftPressed());

}

fileStream.close();

}

LocalFree(szFileFullPath);

}

return VOID();

}

VOID MainDialog::UpdateMenu()

{

HMENU hNewMenu = LoadMenu(

GetModuleHandle(NULL),

MAKEINTRESOURCE(IDR\_MENU\_MAIN)

);

HMENU hOldMenu = GetMenu(\_hwnd);

if (hOldMenu != NULL)

{

DestroyMenu(hOldMenu);

}

if (hNewMenu != NULL)

{

SetMenu(\_hwnd, hNewMenu);

}

HMENU hmLangauge = GetSubMenu(hNewMenu, 1);

\_hmLevel = GetSubMenu(hNewMenu, 2);

\_hmUILanguage = GetSubMenu(hmLangauge, 0);

\_hmKeyboardLanguage = GetSubMenu(hmLangauge, 1);

\_hmEnLevel = GetSubMenu(\_hmLevel, 0);

\_hmRuLevel = GetSubMenu(\_hmLevel, 1);

}

VOID MainDialog::AlignElements()

{

HWND hwndRichEdit = GetDlgItem(\_hwnd, IDC\_RICHEDIT\_TEXT);

SIZE sizeRichEdit = GetClientSize(hwndRichEdit);

SIZE windowSize = GetClientSize();

SIZE sizeKeyboard = \_keyboard.GetSize();

SIZE sizeText = \_text.GetSize();

SIZE sizeStat = \_statistics.GetSize();

WORD offset = 30;

MoveWindow(

\_text.GetWindowHandle(),

windowSize.cx / 2 - \_text.GetClientSize().cx / 2,

\_text.GetLocalPosition().y,

sizeText.cx,

sizeText.cy,

TRUE

);

MoveWindow(

\_keyboard.GetWindowHandle(),

windowSize.cx / 2 - \_keyboard.GetClientSize().cx / 2,

\_keyboard.GetLocalPosition().y,

sizeKeyboard.cx,

sizeKeyboard.cy,

TRUE

);

MoveWindow(

\_statistics.GetWindowHandle(),

\_keyboard.GetLocalPosition().x - 50,

\_text.GetLocalPosition().y + 50,

sizeStat.cx,

sizeStat.cy,

TRUE

);

}

BOOL MainDialog::CheckIfLevel(INT id)

{

if (\_typingLanguage->LoadLevel(id, \_text) != \_T(""))

{

HMENU hMenu = NULL;

switch (\_typingLanguage->Language())

{

case Language::English:

hMenu = \_hmEnLevel;

break;

case Language::Russian:

hMenu = \_hmRuLevel;

break;

default:

break;

}

\_iCurrentLevel = GetMenuItemPosById(hMenu, id) + 1;

\_statistics.Reset();

\_keyboard.Reset();

const TypeButtonInfo \* nextCharInfo = (\*\_typingLanguage)[\_text.CurrentCharacter()];

if (nextCharInfo != NULL)

{

\_keyboard.Select(nextCharInfo->Id(), nextCharInfo->ShiftPressed());

}

return TRUE;

}

return FALSE;

}

VOID MainDialog::OnLevelEnd()

{

KillTimer(\_hwnd, IDT\_TIMER\_SPEED);

UpdateSpeedStat();

\_millisecondsStart = 0;

\_keyboard.Reset();

if (\_statistics.Accuracy() > 80)

{

::MessageBox(

\_hwnd,

\_local.QueryString(IDS\_STRING\_LEVEL\_COMPLETED).c\_str(),

\_TEXT(""),

MB\_OK | MB\_ICONINFORMATION

);

User & currentUser = \_userManagement.GetCurrentUser();

if (currentUser.GetLevel(\_typingLanguage->Language()) == \_iCurrentLevel)

{

currentUser.SetLevel(\_typingLanguage->Language(), \_iCurrentLevel + 1);

}

}

else

{

::MessageBox(

\_hwnd,

\_local.QueryString(IDS\_STRING\_LEVEL\_FAILED).c\_str(),

\_TEXT(""),

MB\_OK | MB\_ICONWARNING

);

}

\_iCurrentLevel = 0;

}

}

TextDialog.h

#pragma once

#include "BaseDialogWindow.h"

#include "stdafx.h"

#include <Richedit.h>

namespace CourseWork\_WinApi

{

class TextDialog

: public BaseDialogWindow

{

public:

TextDialog();

~TextDialog();

virtual INT\_PTR DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

VOID SetText(tString string);

BOOL Wrong();

BOOL Right();

BOOL isEnd();

BOOL Undo();

INT RightSymbolsCount();

INT WrongSymbolsCount();

TCHAR CurrentCharacter();

TCHAR NextCharacter();

INT CursorPos();

VOID Reset();

protected:

BOOL Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);

VOID Select(LONG cpMin, LONG cpMax);

VOID ColorSelection(COLORREF text, COLORREF background);

VOID UnderlineSelection();

VOID ShowCursor();

COLORREF \_unprintedText;

COLORREF \_printedText;

COLORREF \_wrongText;

private:

size\_t \_nCursor;

tString \_text;

std::vector<INT> \_vecWrongSymbolsPos;

/\*resources\*/

HFONT \_hfont;

};

}

TextDialog.cpp

#include "stdafx.h"

#include "TextDialog.h"

namespace CourseWork\_WinApi

{

TextDialog::TextDialog()

:BaseDialogWindow(IDD\_DIALOG\_TEXT)

{

\_nCursor = 0;

\_printedText = RGB(0, 0, 0);

\_wrongText = RGB(239, 83, 80);

\_unprintedText = RGB(147, 147, 147);

}

TextDialog::~TextDialog()

{

DeleteObject(\_hfont);

RemoveFontResource(\_TEXT("DejaVuSansMono.ttf"));

}

INT\_PTR TextDialog::DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

HANDLE\_MSG(hwnd, WM\_INITDIALOG, Cls\_OnInitDialog);

}

return FALSE;

}

VOID TextDialog::SetText(tString string)

{

Select(0, -1);

ColorSelection(\_unprintedText, RGB(255, 255, 255));

SendDlgItemMessage(

\_hwnd,

IDC\_RICHEDIT\_TEXT,

EM\_REPLACESEL,

0,

reinterpret\_cast<LPARAM>(string.c\_str())

);

Select(-1, 0);

\_vecWrongSymbolsPos.clear();

\_nCursor = 0;

ShowCursor();

\_text = string;

}

BOOL TextDialog::Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)

{

EnableWindow(hwnd, FALSE);

/\*Setting font\*/

HWND hRich = GetDlgItem(hwnd, IDC\_RICHEDIT\_TEXT);

LOGFONT LogFont;

::ZeroMemory(&LogFont, sizeof(LOGFONT));

LogFont.lfHeight = \_dpiScale.Scale(44);

LogFont.lfQuality = CLEARTYPE\_QUALITY;

AddFontResource(TEXT("fonts\\DejaVuSansMono.ttf"));

LogFont.lfPitchAndFamily = FIXED\_PITCH | FF\_MODERN;

::lstrcpy(LogFont.lfFaceName, TEXT("DejaVu Sans Mono"));

HFONT \_hfont = ::CreateFontIndirect(&LogFont);

::SendMessage(

hRich,

WM\_SETFONT,

reinterpret\_cast<WPARAM>(\_hfont),

MAKELPARAM(TRUE, NULL)

);

Select(0, -1);

ColorSelection(\_unprintedText, RGB(255, 255, 255));

return 0;

}

VOID TextDialog::Select(LONG cpMin, LONG cpMax)

{

//EnableWindow(\_hwnd, true);

SetFocus(\_hwnd);

CHARRANGE range{ cpMin, cpMax };

HWND hwndEdit = GetDlgItem(\_hwnd, IDC\_RICHEDIT\_TEXT);

//SetFocus(hwndEdit);

SendMessage(

hwndEdit,

EM\_EXSETSEL,

NULL,

reinterpret\_cast<LPARAM>(&range)

);

return void();

}

VOID TextDialog::ColorSelection(COLORREF textColor, COLORREF backgroundColor)

{

CHARFORMAT2 chrfrm;

ZeroMemory(&chrfrm, sizeof(CHARFORMAT2));

chrfrm.cbSize = sizeof(CHARFORMAT2);

chrfrm.dwMask = CFM\_COLOR | CFM\_BACKCOLOR;

chrfrm.crTextColor = textColor;

chrfrm.crBackColor = backgroundColor;

SendDlgItemMessage(

\_hwnd,

IDC\_RICHEDIT\_TEXT,

EM\_SETCHARFORMAT,

SCF\_SELECTION,

(LPARAM)&chrfrm

);

}

VOID TextDialog::UnderlineSelection()

{

CHARFORMAT2 chrfrm;

ZeroMemory(&chrfrm, sizeof(CHARFORMAT2));

chrfrm.cbSize = sizeof(CHARFORMAT2);

chrfrm.dwMask = CFM\_UNDERLINE;

chrfrm.dwEffects = CFE\_UNDERLINE;

SendDlgItemMessage(

\_hwnd,

IDC\_RICHEDIT\_TEXT,

EM\_SETCHARFORMAT,

SCF\_SELECTION,

(LPARAM)&chrfrm

);

}

BOOL TextDialog::Wrong()

{

if (isEnd())

{

return FALSE;

}

Select(\_nCursor, \_nCursor + 1);

\_vecWrongSymbolsPos.push\_back(\_nCursor);

ColorSelection(RGB(255, 255, 255), \_wrongText);

if (CurrentCharacter() == \_T('\u23ce'))

{

\_nCursor++;

}

\_nCursor++;

ShowCursor();

return TRUE;

}

BOOL TextDialog::Right()

{

if (isEnd())

{

return FALSE;

}

Select(\_nCursor, \_nCursor + 1);

ColorSelection(\_printedText, RGB(255, 255, 255));

if (CurrentCharacter() == \_T('\u23ce'))

{

\_nCursor++;

}

\_nCursor++;

ShowCursor();

return TRUE;

}

BOOL TextDialog::isEnd()

{

return \_nCursor > \_text.length() - 1;

}

BOOL TextDialog::Undo()

{

if (\_nCursor > 0)

{

ColorSelection(\_unprintedText, RGB(255, 255, 255));

\_nCursor--;

if (CurrentCharacter() == '\n')

{

\_nCursor--;

}

if (\_vecWrongSymbolsPos.size() > 0 &&

\*(\_vecWrongSymbolsPos.end() - 1) == \_nCursor)

{

\_vecWrongSymbolsPos.pop\_back();

}

ShowCursor();

return TRUE;

}

return FALSE;

}

INT TextDialog::RightSymbolsCount()

{

return \_nCursor - \_vecWrongSymbolsPos.size();

}

INT TextDialog::WrongSymbolsCount()

{

return \_vecWrongSymbolsPos.size();

}

TCHAR TextDialog::CurrentCharacter()

{

if (!isEnd())

{

return \_text[\_nCursor];

}

return \_T('\0');

}

TCHAR TextDialog::NextCharacter()

{

if (\_nCursor < \_text.length() - 1)

{

return \_text[\_nCursor + 1];

}

return \_T('\0');

}

INT TextDialog::CursorPos()

{

return \_nCursor;

}

VOID TextDialog::Reset()

{

Select(0, -1);

ColorSelection(\_unprintedText, RGB(255, 255, 255));

SendDlgItemMessage(

\_hwnd,

IDC\_RICHEDIT\_TEXT,

EM\_REPLACESEL,

0,

reinterpret\_cast<LPARAM>(\_T(""))

);

\_nCursor = 0;

Select(-1, 0);

\_vecWrongSymbolsPos.clear();

\_text.clear();

}

VOID TextDialog::ShowCursor()

{

if (isEnd())

{

return;

}

HWND rich = GetDlgItem(\_hwnd, IDC\_RICHEDIT\_TEXT);

INT line = SendMessage(rich, EM\_LINEFROMCHAR, \_nCursor, NULL);

INT firstVisibleLine = SendMessage(rich, EM\_GETFIRSTVISIBLELINE, NULL, NULL);

if (line - 1 > firstVisibleLine)

{

SendMessage(rich, EM\_LINESCROLL, NULL, line - 1 - firstVisibleLine);

}

Select(\_nCursor, \_nCursor + 1);

ColorSelection(RGB(255, 255, 255), RGB(33, 150, 243));

}

}

UserLogInDialog.h

#pragma once

#include "BaseDialogWindow.h"

#include "UserManagement.h"

#include "stdafx.h"

namespace CourseWork\_WinApi

{

class UserLogInDialog :

public BaseDialogWindow

{

public:

UserLogInDialog(UserManagement & managment);

~UserLogInDialog();

INT\_PTR DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

protected:

tString \_nickName;

tString \_password;

/\*\*/

private:

/\*event handlers\*/

VOID Cls\_OnClose(HWND hwnd);

VOID Cls\_OnDestroy(HWND hwnd);

BOOL Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);

VOID Cls\_OnCommand(HWND hwnd, INT id, HWND hwndCtl, UINT codeNotify);

VOID Cls\_OnDrawItem(HWND hwnd, CONST DRAWITEMSTRUCT \* lpDrawItem);

/\*``````````````````````````````````````````````````````````````\*/

VOID OnLogInButtonClick(HWND hCtl);

VOID OnRegisterButtonClick(HWND hCtl);

VOID OnNickNameEditChange(HWND hCtl);

VOID OnPasswordEditChange(HWND hCtl);

/\*\*/

VOID InitUIText();

VOID OnChangeLanguageButtonClick(INT buttonId);

/\*\*/

UserManagement & \_userManagement;

/\*\*/

size\_t \_iMinNameLength;

size\_t \_iMaxNameLength;

size\_t \_iMinPasswordLength;

size\_t \_iMaxPasswordLength;

HFONT \_hFont;

};

}

UserLogInDialog.cpp

#include "stdafx.h"

#include "UserLogInDialog.h"

#include "resource.h"

#include <strsafe.h>

namespace CourseWork\_WinApi

{

INT\_PTR UserLogInDialog::DlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

HANDLE\_MSG(hwnd, WM\_CLOSE, Cls\_OnClose);

HANDLE\_MSG(hwnd, WM\_DESTROY, Cls\_OnDestroy);

HANDLE\_MSG(hwnd, WM\_INITDIALOG, Cls\_OnInitDialog);

HANDLE\_MSG(hwnd, WM\_COMMAND, Cls\_OnCommand);

HANDLE\_MSG(hwnd, WM\_DRAWITEM, Cls\_OnDrawItem);

}

return false;

}

UserLogInDialog::UserLogInDialog(UserManagement & managment)

:BaseDialogWindow(IDD\_DIALOG\_LOG\_IN),

\_userManagement(managment)

{

\_iMinNameLength = 2;

\_iMaxNameLength = 24;

\_iMinPasswordLength = 4;

\_iMaxPasswordLength = 16;

}

UserLogInDialog::~UserLogInDialog()

{

DeleteObject(\_hFont);

}

VOID UserLogInDialog::Cls\_OnClose(HWND hwnd)

{

Close();

}

void UserLogInDialog::Cls\_OnDestroy(HWND hwnd)

{

}

BOOL UserLogInDialog::Cls\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)

{

HBITMAP hbmpRuFlag = ::LoadBitmap(

::GetModuleHandle(NULL),

MAKEINTRESOURCE(IDB\_BITMAP\_FLAG\_RU)

);

HBITMAP hbmpEnFlag = ::LoadBitmap(

::GetModuleHandle(NULL),

MAKEINTRESOURCE(IDB\_BITMAP\_FLAG\_EN)

);

::SendDlgItemMessage(

hwnd, IDC\_BUTTON\_LANGUAGE\_EN, BM\_SETIMAGE, IMAGE\_BITMAP,

reinterpret\_cast<LPARAM>(hbmpEnFlag)

);

::SendDlgItemMessage(

hwnd, IDC\_BUTTON\_LANGUAGE\_RU, BM\_SETIMAGE, IMAGE\_BITMAP,

reinterpret\_cast<LPARAM>(hbmpRuFlag)

);

/\*Setting font\*/

LOGFONT LogFont;

::ZeroMemory(&LogFont, sizeof(LOGFONT));

LogFont.lfHeight = \_dpiScale.Scale(45);

::StringCchCopy(LogFont.lfFaceName, 32, TEXT("Times New Roman"));

LogFont.lfItalic = TRUE;

\_hFont = ::CreateFontIndirect(&LogFont);

::SendDlgItemMessage(

hwnd,

IDC\_STATIC\_LOG\_IN\_TITLE,

WM\_SETFONT,

reinterpret\_cast<WPARAM>(\_hFont),

MAKELPARAM(TRUE, NULL)

);

/\*\*/

\_local.SetLanguage(Language::Russian);

InitUIText();

return TRUE;

}

void UserLogInDialog::Cls\_OnCommand(HWND hwnd, INT id, HWND hwndCtl, UINT codeNotify)

{

switch (GetDlgCtrlID(hwndCtl))

{

case IDC\_BUTTON\_LOG\_IN:

switch (codeNotify)

{

case BN\_CLICKED:

OnLogInButtonClick(hwndCtl);

break;

}

break;

case IDC\_BUTTON\_REGISTER:

switch (codeNotify)

{

case BN\_CLICKED:

OnRegisterButtonClick(hwndCtl);

break;

}

break;

case IDC\_BUTTON\_LANGUAGE\_EN:

OnChangeLanguageButtonClick(IDC\_BUTTON\_LANGUAGE\_EN);

break;

case IDC\_BUTTON\_LANGUAGE\_RU:

OnChangeLanguageButtonClick(IDC\_BUTTON\_LANGUAGE\_RU);

break;

case IDC\_EDIT\_NAME:

switch (codeNotify)

{

case EN\_CHANGE:

OnNickNameEditChange(hwndCtl);

break;

}

break;

case IDC\_EDIT\_PASSWORD:

switch (codeNotify)

{

case EN\_CHANGE:

OnPasswordEditChange(hwndCtl);

break;

}

break;

}

}

void UserLogInDialog::Cls\_OnDrawItem(HWND hwnd, CONST DRAWITEMSTRUCT \* lpDrawItem)

{

switch (lpDrawItem->CtlType)

{

//ODT\_BUTTON:

case ODT\_BUTTON:

if (lpDrawItem->itemAction == ODA\_DRAWENTIRE)

{

if (lpDrawItem->itemState == ODS\_FOCUS)

{

RECT rect = lpDrawItem->rcItem;

if (lpDrawItem->itemState & ODS\_HOTLIGHT)

SetBkColor(lpDrawItem->hDC, RGB(255, 0, 0));

Ellipse(lpDrawItem->hDC, rect.left, rect.top, rect.right, rect.bottom);

DrawText(lpDrawItem->hDC, TEXT("Log In"), 6, &rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);

}

}

else if (lpDrawItem->itemAction == ODA\_FOCUS)

{

RECT rect = lpDrawItem->rcItem;

if (lpDrawItem->itemState & ODS\_HOTLIGHT)

SetBkColor(lpDrawItem->hDC, RGB(122, 0, 0));

Ellipse(lpDrawItem->hDC, rect.left, rect.top, rect.right, rect.bottom);

DrawText(lpDrawItem->hDC, TEXT("shit"), 6, &rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);

}

else {

::MessageBox(NULL, \_TEXT("chudo"), NULL, MB\_OK);

}

}

}

VOID UserLogInDialog::OnLogInButtonClick(HWND hCtl)

{

HWND hwndEditName = GetDlgItem(\_hwnd, IDC\_EDIT\_NAME);

HWND hwndEditPassword = GetDlgItem(\_hwnd, IDC\_EDIT\_PASSWORD);

tString name = GetText(hwndEditName);

tString password = GetText(hwndEditPassword);

EDITBALLOONTIP tip;

tString title = \_local.QueryString(IDS\_STRING\_WARNING);

tip.cbStruct = sizeof(EDITBALLOONTIP);

tip.pszText = NULL;

tip.pszTitle = title.c\_str();

tip.ttiIcon = TTI\_WARNING;

if (name.length() == 0 && password.length() == 0)

{

MessageBox(

\_hwnd,

\_local.QueryString(IDS\_STRING\_EMPTY\_FIELDS).c\_str(),

title.c\_str(),

MB\_OK | MB\_ICONWARNING

);

return;

}

else if (name.length() < \_iMinNameLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_SHORT);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditName, &tip);

}

else if (name.length() > \_iMaxNameLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_LONG);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditName, &tip);

}

else if (password.length() < \_iMinPasswordLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_SHORT);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditPassword, &tip);

}

else if (password.length() > \_iMaxPasswordLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_LONG);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditPassword, &tip);

}

else if (!\_userManagement.IsUserExists(\_nickName))

{

tString text = \_local.QueryString(IDS\_STRING\_USER\_DOES\_NOT\_EXIST);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditName, &tip);

}

else if (\_userManagement.IsUserExists(\_nickName) &&

\_userManagement.GetUserByNickname(\_nickName)->GetPassword() != password)

{

tString text = \_local.QueryString(IDS\_STRING\_INVALID\_PASSWORD);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditPassword, &tip);

}

else

{

if (\_userManagement.LogIn(name, password) == LoginResult::SUCCESS)

{

\_nResult = IDOK;

Close();

}

else

{

//TODO: insert message box here

}

}

}

VOID UserLogInDialog::OnRegisterButtonClick(HWND hCtl)

{

HWND hwndEditName = GetDlgItem(\_hwnd, IDC\_EDIT\_NAME);

HWND hwndEditPassword = GetDlgItem(\_hwnd, IDC\_EDIT\_PASSWORD);

tString name = GetText(hwndEditName);

tString password = GetText(hwndEditPassword);

EDITBALLOONTIP tip;

tString title = \_local.QueryString(IDS\_STRING\_WARNING);

tip.cbStruct = sizeof(EDITBALLOONTIP);

tip.pszText = NULL;

tip.pszTitle = title.c\_str();

tip.ttiIcon = TTI\_WARNING;

if (name.length() == 0 && password.length() == 0)

{

MessageBox(

\_hwnd,

\_local.QueryString(IDS\_STRING\_EMPTY\_FIELDS).c\_str(),

title.c\_str(),

MB\_OK | MB\_ICONWARNING

);

return;

}

else if (name.length() < \_iMinNameLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_SHORT);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditName, &tip);

}

else if (name.length() > \_iMaxNameLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_LONG);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditName, &tip);

}

else if (password.length() < \_iMinPasswordLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_SHORT);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditPassword, &tip);

}

else if (password.length() > \_iMaxPasswordLength)

{

tString text = \_local.QueryString(IDS\_STRING\_TOO\_LONG);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditPassword, &tip);

}

else

{

User \* newUser = \_userManagement.NewUser(name, password);

if (newUser != nullptr)

{

\_userManagement.LogIn(\*newUser);

\_nResult = IDOK;

Close();

}

else

{

tString text = \_local.QueryString(IDS\_STRING\_USED\_NAME);

tip.pszText = text.c\_str();

Edit\_ShowBalloonTip(hwndEditName, &tip);

}

}

}

VOID UserLogInDialog::OnNickNameEditChange(HWND hCtl)

{

\_nickName = GetText(hCtl);

}

VOID UserLogInDialog::OnPasswordEditChange(HWND hCtl)

{

\_password = GetText(hCtl);

}

VOID UserLogInDialog::OnChangeLanguageButtonClick(INT buttonId)

{

switch (buttonId)

{

case IDC\_BUTTON\_LANGUAGE\_EN:

if (\_local.GetLanguage() == Language::English)

{

break;

}

\_local.SetLanguage(Language::English);

InitUIText();

break;

case IDC\_BUTTON\_LANGUAGE\_RU:

if (\_local.GetLanguage() == Language::Russian)

{

break;

}

\_local.SetLanguage(Language::Russian);

InitUIText();

break;

default:

break;

}

}

VOID UserLogInDialog::InitUIText()

{

HWND hUserName = GetDlgItem(\_hwnd, IDC\_EDIT\_NAME);

HWND hPassword = GetDlgItem(\_hwnd, IDC\_EDIT\_PASSWORD);

HWND hLogInButton = GetDlgItem(\_hwnd, IDC\_BUTTON\_LOG\_IN);

HWND hRegisterButton = GetDlgItem(\_hwnd, IDC\_BUTTON\_REGISTER);

Edit\_SetCueBannerText(hUserName, \_local.QueryString(IDS\_STRING\_USER\_NAME).c\_str());

Edit\_LimitText(hUserName, 24);

Edit\_SetCueBannerText(hPassword, \_local.QueryString(IDS\_STRING\_PASSWORD).c\_str());

Edit\_LimitText(hPassword, 12);

SetWindowText(hLogInButton, \_local.QueryString(IDS\_STRING\_LOG\_IN).c\_str());

SetWindowText(hRegisterButton, \_local.QueryString(IDS\_STRING\_REGISTER).c\_str());

SetDlgItemText(\_hwnd, IDC\_STATIC\_LOG\_IN\_TITLE, \_local.QueryString(IDS\_STRING\_TITLE).c\_str());

}

}

BaseTypingLanuage.h

#pragma once

#include <vector>

#include "TypeButtonInfo.h"

#include "BaseDialogWindow.h"

#include "TextDialog.h"

#include "KeyboardDialog.h"

#include "GameStatisticsDialog.h"

namespace CourseWork\_WinApi

{

namespace Shift

{

enum Shift

{

NO, YES

};

}

class BaseTypingLanguage

{

public:

const TypeButtonInfo \* operator[](TCHAR character);

VOID InitKeyBoard(KeyboardDialog & keyboard);

tString LoadLevel(INT id\_level, TextDialog & textDialog);

tString GetLevelText(INT id\_level);

virtual Language::Language Language() = 0;

protected:

VOID AddButton(UINT id, TCHAR character, Shift::Shift shift, BOOL bShow = TRUE);

VOID BaseInit();

virtual VOID Initialize() = 0;

std::map<INT, tString> \_levels;

private:

std::vector<TypeButtonInfo> \_buttonInfo;

};

}

BaseTypingLanguage.cpp

#include "stdafx.h"

#include "BaseTypingLanguage.h"

namespace CourseWork\_WinApi

{

const TypeButtonInfo \* BaseTypingLanguage::operator[](TCHAR character)

{

std::vector<TypeButtonInfo>::iterator it;

it = std::find\_if(\_buttonInfo.begin(), \_buttonInfo.end(), [character](TypeButtonInfo & info) {

return info.Character() == character;

});

if (it != \_buttonInfo.end())

{

return &(\*it);

}

return NULL;

}

VOID BaseTypingLanguage::InitKeyBoard(KeyboardDialog & keyboard)

{

HWND hKeyboard = keyboard.GetWindowHandle();

keyboard.Reset();

::SendMessage(hKeyboard, WM\_SETREDRAW, FALSE, 0);

for (auto it = \_buttonInfo.begin(); it != \_buttonInfo.end(); it++)

{

if (it->ShiftPressed() && it->Show())

{

tString txt(1, it->Character());

::SetDlgItemText(hKeyboard, it->Id(), txt.c\_str());

}

}

for (auto it = \_buttonInfo.begin(); it != \_buttonInfo.end(); it++)

{

if (!it->ShiftPressed() && it->Show())

{

HWND hButton = GetDlgItem(hKeyboard, it->Id());

tString txt = BaseDialogWindow::GetText(hButton);

txt += TEXT("\n");

txt += it->Character();

::SetWindowText(hButton, txt.c\_str());

}

}

SendMessage(hKeyboard, WM\_SETREDRAW, TRUE, 0);

RedrawWindow(

hKeyboard, NULL, NULL,

RDW\_ERASE | RDW\_FRAME | RDW\_INVALIDATE | RDW\_ALLCHILDREN

);

}

tString BaseTypingLanguage::LoadLevel( INT id\_level, TextDialog & textDialog)

{

tString str = \_levels[id\_level];

textDialog.SetText(str);

return str;

}

tString BaseTypingLanguage::GetLevelText(INT id\_level)

{

return \_levels[id\_level];

}

VOID BaseTypingLanguage::AddButton(UINT id, TCHAR character, Shift::Shift shift, BOOL bShow)

{

\_buttonInfo.push\_back(TypeButtonInfo(id, character, shift == Shift::YES ? TRUE : FALSE, bShow));

}

VOID BaseTypingLanguage::BaseInit()

{

AddButton(IDC\_BUTTON\_SPACE, \_T(' '), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_ENTER, \_T('\u23ce'), Shift::NO, FALSE);

//AddButton(IDC\_BUTTON\_BACKSPACE, \_T('\b'), Shift::NO, FALSE);

}

}

EnglishTypingLanguage.h

#pragma once

#include "BaseTypingLanguage.h"

namespace CourseWork\_WinApi

{

class EnglishTypingLanguage :

public BaseTypingLanguage

{

public:

EnglishTypingLanguage();

~EnglishTypingLanguage();

protected:

// Inherited via BaseTypingLanguage

virtual VOID Initialize() override;

// Inherited via BaseTypingLanguage

virtual Language::Language Language() override;

};

}

EnglishTypingLanguage.cpp

#include "stdafx.h"

#include "EnglishTypingLanguage.h"

namespace CourseWork\_WinApi

{

EnglishTypingLanguage::EnglishTypingLanguage()

{

Initialize();

\_levels[ID\_ENGLISH\_LEVEL\_EN\_FJ] = \_T("ffffffjjjjjjjjjj");

\_levels[ID\_ENGLISH\_LEVEL\_EN\_SPACEBAR] = \_T("jjjjjj fj fj fj fj fffff jjjjj jj jj ff ff jf jf jf");

}

EnglishTypingLanguage::~EnglishTypingLanguage()

{

}

VOID EnglishTypingLanguage::Initialize()

{

BaseInit();

AddButton(IDC\_BUTTON\_BACK\_QUOTE, \_T('~'), Shift::YES);

AddButton(IDC\_BUTTON\_BACK\_QUOTE, \_T('`'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_1, \_T('!'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_1, \_T('1'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_2, \_T('@'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_2, \_T('2'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_3, \_T('#'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_3, \_T('3'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_4, \_T('$'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_4, \_T('4'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_5, \_T('%'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_5, \_T('5'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_6, \_T('^'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_6, \_T('6'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_7, \_T('&'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_7, \_T('7'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_8, \_T('\*'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_8, \_T('8'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_9, \_T('('), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_9, \_T('9'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_0, \_T(')'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_0, \_T('0'), Shift::NO);

AddButton(IDC\_BUTTON\_MINUS, \_T('\_'), Shift::YES);

AddButton(IDC\_BUTTON\_MINUS, \_T('-'), Shift::NO);

AddButton(IDC\_BUTTON\_EQUAL, \_T('+'), Shift::YES);

AddButton(IDC\_BUTTON\_EQUAL, \_T('='), Shift::NO);

AddButton(IDC\_BUTTON\_KEY\_Q, \_T('Q'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_Q, \_T('q'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_W, \_T('W'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_W, \_T('w'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_E, \_T('E'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_E, \_T('e'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_R, \_T('R'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_R, \_T('r'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_T, \_T('T'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_T, \_T('t'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_Y, \_T('Y'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_Y, \_T('y'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_U, \_T('U'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_U, \_T('u'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_I, \_T('I'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_I, \_T('i'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_O, \_T('O'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_O, \_T('o'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_P, \_T('P'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_P, \_T('p'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_BRACKET\_LEFT, \_T('{'), Shift::YES);

AddButton(IDC\_BUTTON\_BRACKET\_LEFT, \_T('['), Shift::NO);

AddButton(IDC\_BUTTON\_BRACKET\_RIGHT, \_T('}'), Shift::YES);

AddButton(IDC\_BUTTON\_BRACKET\_RIGHT, \_T(']'), Shift::NO);

AddButton(IDC\_BUTTON\_BACKSLASH, \_T('|'), Shift::YES);

AddButton(IDC\_BUTTON\_BACKSLASH, \_T('\\'), Shift::NO);

AddButton(IDC\_BUTTON\_KEY\_A, \_T('A'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_A, \_T('a'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_S, \_T('S'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_S, \_T('s'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_D, \_T('D'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_D, \_T('d'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_F, \_T('F'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_F, \_T('f'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_G, \_T('G'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_G, \_T('g'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_H, \_T('H'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_H, \_T('h'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_J, \_T('J'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_J, \_T('j'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_K, \_T('K'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_K, \_T('k'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_L, \_T('L'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_L, \_T('l'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_SEMICOLON, \_T(':'), Shift::YES);

AddButton(IDC\_BUTTON\_SEMICOLON, \_T(';'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_QUOTE, \_T('"'), Shift::YES);

AddButton(IDC\_BUTTON\_QUOTE, \_T('\''), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_Z, \_T('Z'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_Z, \_T('z'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_X, \_T('X'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_X, \_T('x'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_C, \_T('C'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_C, \_T('c'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_V, \_T('V'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_V, \_T('v'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_B, \_T('B'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_B, \_T('b'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_N, \_T('N'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_N, \_T('n'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_M, \_T('M'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_M, \_T('m'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_COMMA, \_T('<'), Shift::YES);

AddButton(IDC\_BUTTON\_COMMA, \_T(','), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_PERIOD, \_T('<'), Shift::YES);

AddButton(IDC\_BUTTON\_PERIOD, \_T(','), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_SLASH, \_T('?'), Shift::YES);

AddButton(IDC\_BUTTON\_SLASH, \_T('/'), Shift::NO, FALSE);

}

Language::Language EnglishTypingLanguage::Language()

{

return Language::English;

}

}

RussianTypingLanguage.h

#pragma once

#include "BaseTypingLanguage.h"

namespace CourseWork\_WinApi

{

class RussianTypingLanguage :

public BaseTypingLanguage

{

public:

RussianTypingLanguage();

// Inherited via BaseTypingLanguage

protected:

virtual VOID Initialize() override;

// Inherited via BaseTypingLanguage

virtual Language::Language Language() override;

};

}

RussianTypingLanguage.cpp

#include "stdafx.h"

#include "RussianTypingLanguage.h"

namespace CourseWork\_WinApi

{

RussianTypingLanguage::RussianTypingLanguage()

{

Initialize();

\_levels[ID\_RUSSIAN\_LEVEL\_AO] = \_T("аааааааоооооооо");

}

VOID RussianTypingLanguage::Initialize()

{

BaseInit();

AddButton(IDC\_BUTTON\_BACK\_QUOTE, \_T('~'), Shift::YES);

AddButton(IDC\_BUTTON\_BACK\_QUOTE, \_T('`'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_1, \_T('!'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_1, \_T('1'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_2, \_T('"'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_2, \_T('2'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_3, \_T('№'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_3, \_T('3'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_4, \_T(';'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_4, \_T('4'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_5, \_T('%'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_5, \_T('5'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_6, \_T(':'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_6, \_T('6'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_7, \_T('?'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_7, \_T('7'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_8, \_T('\*'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_8, \_T('8'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_9, \_T('('), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_9, \_T('9'), Shift::NO);

AddButton(IDC\_BUTTON\_DIGIT\_0, \_T(')'), Shift::YES);

AddButton(IDC\_BUTTON\_DIGIT\_0, \_T('0'), Shift::NO);

AddButton(IDC\_BUTTON\_MINUS, \_T('\_'), Shift::YES);

AddButton(IDC\_BUTTON\_MINUS, \_T('-'), Shift::NO);

AddButton(IDC\_BUTTON\_EQUAL, \_T('+'), Shift::YES);

AddButton(IDC\_BUTTON\_EQUAL, \_T('='), Shift::NO);

AddButton(IDC\_BUTTON\_KEY\_Q, \_T('Й'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_Q, \_T('й'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_W, \_T('Ц'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_W, \_T('ц'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_E, \_T('У'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_E, \_T('у'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_R, \_T('К'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_R, \_T('к'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_T, \_T('Е'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_T, \_T('е'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_Y, \_T('Н'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_Y, \_T('н'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_U, \_T('Г'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_U, \_T('г'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_I, \_T('Ш'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_I, \_T('ш'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_O, \_T('Щ'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_O, \_T('щ'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_P, \_T('З'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_P, \_T('з'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_BRACKET\_LEFT, \_T('Х'), Shift::YES);

AddButton(IDC\_BUTTON\_BRACKET\_LEFT, \_T('х'), Shift::NO);

AddButton(IDC\_BUTTON\_BRACKET\_RIGHT, \_T('Ъ'), Shift::YES);

AddButton(IDC\_BUTTON\_BRACKET\_RIGHT, \_T('ъ'), Shift::NO);

AddButton(IDC\_BUTTON\_BACKSLASH, \_T('/'), Shift::YES);

AddButton(IDC\_BUTTON\_BACKSLASH, \_T('\\'), Shift::NO);

AddButton(IDC\_BUTTON\_KEY\_A, \_T('Ф'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_A, \_T('ф'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_S, \_T('Ы'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_S, \_T('ы'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_D, \_T('В'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_D, \_T('в'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_F, \_T('А'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_F, \_T('а'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_G, \_T('П'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_G, \_T('п'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_H, \_T('Р'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_H, \_T('р'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_J, \_T('О'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_J, \_T('о'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_K, \_T('Л'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_K, \_T('л'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_L, \_T('Д'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_L, \_T('д'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_SEMICOLON, \_T('Ж'), Shift::YES);

AddButton(IDC\_BUTTON\_SEMICOLON, \_T('ж'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_QUOTE, \_T('Э'), Shift::YES);

AddButton(IDC\_BUTTON\_QUOTE, \_T('э'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_Z, \_T('Я'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_Z, \_T('я'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_X, \_T('Ч'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_X, \_T('ч'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_C, \_T('С'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_C, \_T('с'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_V, \_T('М'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_V, \_T('м'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_B, \_T('И'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_B, \_T('и'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_N, \_T('Т'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_N, \_T('т'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_KEY\_M, \_T('Ь'), Shift::YES);

AddButton(IDC\_BUTTON\_KEY\_M, \_T('ь'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_COMMA, \_T('Б'), Shift::YES);

AddButton(IDC\_BUTTON\_COMMA, \_T('б'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_PERIOD, \_T('Ю'), Shift::YES);

AddButton(IDC\_BUTTON\_PERIOD, \_T('ю'), Shift::NO, FALSE);

AddButton(IDC\_BUTTON\_SLASH, \_T('.'), Shift::NO);

AddButton(IDC\_BUTTON\_SLASH, \_T(','), Shift::YES, FALSE);

}

Language::Language RussianTypingLanguage::Language()

{

return Language::Russian;

}

}

LocalizationObject.h

#pragma once

#include "stdafx.h"

namespace CourseWork\_WinApi

{

namespace Language

{

enum Language

{

English, Russian

};

}

class LocalizationObject

{

public:

LocalizationObject();

~LocalizationObject();

Language::Language GetLanguage();

VOID SetLanguage(Language::Language language);

tString QueryString(UINT identifier);

protected:

Language::Language \_language;

private:

};

}

LocalizationObject.cpp

#include "stdafx.h"

#include "LocalizationObject.h"

namespace CourseWork\_WinApi

{

LocalizationObject::LocalizationObject()

{

}

LocalizationObject::~LocalizationObject()

{

}

Language::Language LocalizationObject::GetLanguage()

{

return \_language;

}

VOID LocalizationObject::SetLanguage(Language::Language language)

{

\_language = language;

switch (language)

{

case Language::English:

::SetThreadUILanguage(

MAKELCID(LANG\_ENGLISH, SUBLANG\_NEUTRAL)

);

break;

case Language::Russian:

::SetThreadUILanguage(

MAKELCID(LANG\_RUSSIAN, SUBLANG\_NEUTRAL)

);

break;

}

}

tString CourseWork\_WinApi::LocalizationObject::QueryString(UINT identifier)

{

LPTSTR p = nullptr;

int len = ::LoadString(NULL, identifier, reinterpret\_cast<LPTSTR>(&p), 0);

if (len > 0)

{

return tString(p, static\_cast<size\_t>(len));

}

return tString();

}

}

TypeButtonInfo.h

#pragma once

#include <Windows.h>

#include <tchar.h>

namespace CourseWork\_WinApi

{

class TypeButtonInfo

{

public:

TypeButtonInfo(UINT uButtonID, TCHAR tchCharacter, BOOL bShiftPrssed, BOOL \_bShow);

~TypeButtonInfo();

UINT Id() const;

TCHAR Character() const;

BOOL ShiftPressed() const;

BOOL Show() const;

private:

UINT \_uButtonID;

TCHAR \_tchCaracter;

BOOL \_bShift;

BOOL \_bShow;

};

}

TypeButtonInfo.cpp

#include "stdafx.h"

#include "TypeButtonInfo.h"

namespace CourseWork\_WinApi

{

TypeButtonInfo::TypeButtonInfo(

UINT uButtonID,

TCHAR tchCharacter,

BOOL bShiftPrssed,

BOOL bShow)

{

\_uButtonID = uButtonID;

\_tchCaracter = tchCharacter;

\_bShift = bShiftPrssed;

\_bShow = bShow;

}

TypeButtonInfo::~TypeButtonInfo()

{

}

UINT TypeButtonInfo::Id() const

{

return \_uButtonID;

}

TCHAR TypeButtonInfo::Character() const

{

return \_tchCaracter;

}

BOOL TypeButtonInfo::ShiftPressed() const

{

return \_bShift;

}

BOOL TypeButtonInfo::Show() const

{

return \_bShow;

}

}

User.h

#pragma once

#include "stdafx.h"

#include "LocalizationObject.h"

namespace CourseWork\_WinApi

{

class User

{

public:

User(tString name, tString password);

User( const User & user);

User(User && user);

User & operator=(const User & user);

User & operator=(User && user);

BOOL operator==(const User & user);

tString GetNickname() const;

VOID SetName(tString name);

tString GetPassword() const;

VOID SetPassword(tString password);

User & SetLevel(Language::Language language, int level);

INT GetLevel(Language::Language langauge);

~User();

private:

tString \_nickname;

tString \_password;

Language::Language \_Interface;

std::map<Language::Language, INT> \_levelMap;

};

}

User.cpp

#include "stdafx.h"

#include "User.h"

namespace CourseWork\_WinApi

{

User::User(tString name, tString password)

{

SetName(name);

SetPassword(password);

}

User::User(const User & user)

{

\_nickname = user.\_nickname;

\_password = user.\_password;

\_levelMap = user.\_levelMap;

}

User::User(User && user)

{

\_nickname = std::move(user.\_nickname);

\_password = std::move(user.\_password);

\_levelMap = std::move(user.\_levelMap);

}

User & User::operator=(const User & user)

{

if (&user != this)

{

\_nickname = user.\_nickname;

\_password = user.\_password;

\_levelMap = user.\_levelMap;

}

return \*this;

}

User & User::operator=(User && user)

{

\_nickname = std::move(user.\_nickname);

\_password = std::move(user.\_password);

\_levelMap = std::move(user.\_levelMap);

return \*this;

}

BOOL User::operator==(const User & user)

{

return this->\_nickname == user.\_nickname && this->\_password == user.\_password;

}

tString User::GetNickname() const

{

return \_nickname;

}

void User::SetName(tString name)

{

\_nickname = name;

}

tString User::GetPassword() const

{

return \_password;

}

void User::SetPassword(tString password)

{

\_password = password;

}

User & User::SetLevel(Language::Language language, int level)

{

\_levelMap[language] = level;

return \*this;

}

INT User::GetLevel(Language::Language language)

{

return \_levelMap[language];

}

User::~User()

{

}

}

UserManagement.h

#pragma once

#include "stdafx.h"

#include "User.h"

#include "CustomExceptions.hpp"

#include <exception>

namespace CourseWork\_WinApi

{

enum LoginResult

{

SUCCESS, INVALIDUSER, INVALIDPASSWORD

};

class UserManagement

{

public:

UserManagement();

User \* NewUser(tString userNickname, tString userPassword);

User \* GetUserByNickname(tString userNickname);

const User \* GetUserByNickname(tString userNickname) const;

BOOL IsUserExists(tString userNickname);

LoginResult LogIn(tString userNickname, tString userPassword);

LoginResult LogIn(const User & user);

~UserManagement();

User & GetCurrentUser();

private:

std::vector<User>::iterator FindUser(tString nickName);

BOOL ReadUser(LPBYTE buffer, DWORD dwSize, DWORD dwFrom, DWORD & dwNumberOfBytesRead);

BOOL SafeUser(HANDLE hFile, User & user);

std::vector<User> \_users;

INT \_nCurrentUser;

};

}

UserManagemetn.cpp

#include "stdafx.h"

#include "UserManagement.h"

#include <Strsafe.h>

namespace CourseWork\_WinApi

{

UserManagement::UserManagement()

{

\_nCurrentUser = -1;

LARGE\_INTEGER fileSize;

OFSTRUCT ofstruct;

ofstruct.cBytes = sizeof(OFSTRUCT);

HANDLE hFile = CreateFile(

TEXT("users"), // name of the write

GENERIC\_READ, // open for writing

0, // do not share

NULL, // default security

OPEN\_ALWAYS, // create new file only

FILE\_ATTRIBUTE\_NORMAL, // normal file

NULL); // no attr. template

if (hFile == INVALID\_HANDLE\_VALUE)

{

return;

}

GetFileSizeEx(hFile, &fileSize);

if (fileSize.QuadPart == 0)

{

CloseHandle(hFile);

return;

}

LPBYTE buffer = new BYTE[fileSize.LowPart];

DWORD bytesRead = 0;

DWORD from = 0;

ReadFile(hFile, buffer, fileSize.LowPart, &bytesRead, NULL);

while (ReadUser(buffer, fileSize.LowPart, from, bytesRead))

{

from += bytesRead;

}

CloseHandle(hFile);

delete[] buffer;

}

User \* UserManagement::NewUser(tString userName, tString userPassword)

{

if (GetUserByNickname(userName) == nullptr)

{

User user(userName, userPassword);

user.SetLevel(Language::English, 1).SetLevel(Language::Russian, 1);

\_users.push\_back(user);

return &(\_users[\_users.size() - 1]);

}

return nullptr;

}

User \* UserManagement::GetUserByNickname(tString nickName)

{

std::vector<User>::iterator it = FindUser(nickName);

//check if something is found return the value;

if (it != \_users.end())

{

return &(\*it);

}

//otherwise return nullptr

return nullptr;

}

const User \* UserManagement::GetUserByNickname(tString userNickname) const

{

return const\_cast<UserManagement\*>(this)->GetUserByNickname(userNickname);

}

BOOL UserManagement::IsUserExists(tString userNickname)

{

return (GetUserByNickname(userNickname) != nullptr);

}

LoginResult UserManagement::LogIn(tString userNickName, tString userPassword)

{

std::vector<User>::iterator it = FindUser(userNickName);

if (it != \_users.end())

{

User & user = (\*it);

if (user.GetPassword() != userPassword)

{

return LoginResult::INVALIDPASSWORD;

}

\_nCurrentUser = it - \_users.begin();

return LoginResult::SUCCESS;

}

return LoginResult::INVALIDUSER;

}

LoginResult UserManagement::LogIn(const User & user)

{

std::vector<User>::iterator it = std::find(\_users.begin(), \_users.end(), user);

if (it != \_users.end())

{

\_nCurrentUser = it - \_users.begin();

return LoginResult::SUCCESS;

}

return LoginResult::INVALIDUSER;

}

UserManagement::~UserManagement()

{

HANDLE hFile = CreateFile(

\_T("users"), // name of the write

GENERIC\_WRITE, // open for writing

0, // do not share

NULL, // default security

CREATE\_ALWAYS, // create new file only

FILE\_ATTRIBUTE\_NORMAL, // normal file

NULL

);

if (hFile == INVALID\_HANDLE\_VALUE)

{

return;

}

for (auto user : \_users)

{

SafeUser(hFile, user);

}

CloseHandle(hFile);

}

User & UserManagement::GetCurrentUser()

{

if (\_nCurrentUser == -1)

{

throw cex::NoCurrentUserException();

}

return \_users[\_nCurrentUser];

}

std::vector<User>::iterator UserManagement::FindUser(tString nickName)

{

return std::find\_if(

\_users.begin(), \_users.end(),

[&nickName](const User & user) { return user.GetNickname() == nickName; }

);

}

BOOL UserManagement::ReadUser(LPBYTE buffer, DWORD size, DWORD from, DWORD & numberOfBytesRead)

{

if (from >= size)

{

return FALSE;

}

LPBYTE lpBufferEnd = buffer + size;

LPTSTR name;

INT nameLen;

LPTSTR surname;

INT surnameLen;

INT iLanguagesCount;

LPTSTR str = reinterpret\_cast<LPTSTR>(buffer + from);

nameLen = ::lstrlen(str) + 1;

if (reinterpret\_cast<LPBYTE>(str + nameLen) > lpBufferEnd)

{

numberOfBytesRead = 0;

return FALSE;

}

name = new TCHAR[nameLen];

::StringCchCopy(name, nameLen, str);

str += nameLen;

surnameLen = ::lstrlen(str) + 1;

if (reinterpret\_cast<LPBYTE>(str + surnameLen) > lpBufferEnd)

{

numberOfBytesRead = 0;

return FALSE;

}

surname = new TCHAR(surnameLen);

::StringCchCopy(surname, surnameLen, str);

str += surnameLen;

User user = User(tString(name), tString(surname));

LPBYTE buffer2 = reinterpret\_cast<LPBYTE>(str);

if (buffer2 + sizeof(INT) > lpBufferEnd)

{

numberOfBytesRead = buffer2 - (buffer + from);

\_users.push\_back(user);

return TRUE;

}

::CopyMemory(&iLanguagesCount, buffer2, sizeof(INT));

buffer2 += sizeof(INT);

Language::Language language;

INT level;

for (INT i = 0; i < iLanguagesCount; i++)

{

if (buffer2 + sizeof(INT) \* 2 > lpBufferEnd)

{

break;

}

::CopyMemory(&language, buffer2, sizeof(INT));

buffer2 += sizeof(INT);

::CopyMemory(&level, buffer2, sizeof(INT));

buffer2 += sizeof(INT);

user.SetLevel(language, level);

}

numberOfBytesRead = buffer2 - (buffer + from);

\_users.push\_back(user);

return TRUE;

}

BOOL UserManagement::SafeUser(HANDLE hFile, User & user)

{

DWORD dwBytesWritten;

BOOL bResult;

INT iNumberOfLanguages = 2;

//writing name

bResult = WriteFile(

hFile,

user.GetNickname().c\_str(),

(user.GetNickname().length() + 1) \* sizeof(TCHAR),

&dwBytesWritten,

NULL

);

if (!bResult)

{

return FALSE;

}

//writing password

bResult = WriteFile(

hFile,

user.GetPassword().c\_str(),

(user.GetPassword().length() + 1) \* sizeof(TCHAR),

&dwBytesWritten,

NULL

);

if (!bResult)

{

return FALSE;

}

//writing count of learning languages

bResult = WriteFile(

hFile,

&iNumberOfLanguages,

sizeof(INT),

&dwBytesWritten,

NULL

);

if (!bResult)

{

return FALSE;

}

Language::Language language;

INT level;

//writing english language result

language = Language::English;

level = user.GetLevel(language);

WriteFile(

hFile,

&language,

sizeof(INT),

&dwBytesWritten,

NULL

);

WriteFile(

hFile,

&level,

sizeof(INT),

&dwBytesWritten,

NULL

);

language = Language::Russian;

level = user.GetLevel(language);

//writing russian language result

WriteFile(

hFile,

&language,

sizeof(INT),

&dwBytesWritten,

NULL

);

WriteFile(

hFile,

&level,

sizeof(INT),

&dwBytesWritten,

NULL

);

return TRUE;

}

}

Main.cpp

#include "stdafx.h"

#include "resource.h"

#include "MainDialog.h"

#include "UserLogInDialog.h"

#include <CommCtrl.h>

#pragma comment(lib, "comctl32")

void inline EnableCommonControls();

INT WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, INT nCmdShow)

{

EnableCommonControls();

auto mainDialog = new CourseWork\_WinApi::MainDialog();

HWND hMainWindow = mainDialog->Create();

MSG msg;

while (GetMessage(&msg, NULL, NULL, NULL))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

delete mainDialog;

return NULL;

}

void inline EnableCommonControls()

{

INITCOMMONCONTROLSEX icc = { sizeof(INITCOMMONCONTROLSEX), ICC\_WIN95\_CLASSES };

InitCommonControlsEx(&icc);

LoadLibrary(TEXT("RICHED20.DLL"));

}