|  |
| --- |
| **МИНОБРНАУКИ РОССИИ**  федеральное государственное бюджетное образовательное  учреждение высшего образования  **«Национальный исследовательский университет «МЭИ»** |

|  |  |
| --- | --- |
| **Институт** | ИВТИ |
| **Кафедра** | ПМИИ |

|  |  |
| --- | --- |
|  |  |
| **Предмет: «Системное программирование»**  **Отчет по лабораторной работе №3**  **Выполнил: студент группы А-13а-19**  **Рамазанов Н. М.**  **Преподаватель: Меньшикова К. Г.**  **Москва, 2021** | |

**1. Краткое описание функций, параметров и используемых событий.**

1) Функции для работы с окнами.

* EnableWindow

BOOL EnableWindow (HWND hWnd, BOOL bEnable);

Функция EnableWindow включает в работу или отключает мышь и ввод с клавиатуры в заданное окно или орган управления. Когда ввод данных заблокирован, окно не принимает ввода щелчков мыши и нажатий клавиши. Когда ввод данных разрешается, окно принимает весь ввод данных.

Параметры: hWnd – дескриптор окна, которое нужно включить / отключить; bEnable – показывает, отключаем или включаем окно.

Если окно было предварительно заблокировано, возвращаемое значение – не нуль, если не было – нуль.

* IsWindowVisible

BOOL IsWindowVisible (HWND hWnd);

Функция IsWindowVisible извлекает данные о состоянии видимости заданного окна.

Параметр: hWnd – дескриптор проверяемого окна.

Если заданное окно, его родительское окно, родители родительского окна и так далее, имеют стиль WS\_VISIBLE, возвращаемое значение не нуль. Иначе, возвращаемое значение нулевое.

* ShowWindow

BOOL EnableWindow (HWND hWnd, int nCmdShow);

Функция ShowWindow устанавливает состояние показа определяемого окна.

Параметры: hWnd – дескриптор окна; nCmdShow – определяет, как окно должно быть показано.

Второй параметр игнорируется при первом вызове приложения ShowWindow, если программа, запустившая приложение, предоставляет структуру STARTUPINFO. В противном случае при первом вызове ShowWindow значение должно быть значением, полученным функцией WinMain в ее параметре nCmdShow. В дальнейшем может принимать значения, указанные в таблице:

|  |  |
| --- | --- |
| **Значение** | **Смысл** |
| **SW\_HIDE** | Скрывает окно и активирует другое окно. |
| **SW\_MAXIMIZE** | Развертывает указанное окно. |
| **SW\_MINIMIZE** | Сворачивает указанное окно и активирует следующее окно верхнего уровня в Z-порядке. |
| **SW\_RESTORE** | Активирует и отображает окно. Если окно свернуто или развернуто, система восстанавливает его исходный размер и положение. Приложение должно указать этот флаг при восстановлении свернутого окна. |
| **SW\_SHOW** | Активирует окно и отображает его в текущем размере и положении. |
| **SW\_SHOWDEFAULT** | Устанавливает состояние показа на основе значения SW\_, указанного в структуре STARTUPINFO, переданной в функцию CreateProcess программой, запустившей приложение. |
| **SW\_SHOWMAXIMIZED** | Активирует окно и отображает его как развернутое окно. |
| **SW\_SHOWMINIMIZED** | Активирует окно и отображает его как свернутое окно. |
| **SW\_SHOWMINNOACTIVE** | Отображает окно в виде свернутого окна, но окно не активируется. |
| **SW\_SHOWNA** | Отображает окно в его текущем размере и положении, но окно не активируется. |
| **SW\_SHOWNOACTIVATE** | Отображает окно в его самом последнем размере и положении, но окно не активируется. |
| **SW\_SHOWNORMAL** | Активирует и отображает окно. Если окно свернуто или развернуто, система восстанавливает его исходный размер и положение. Приложение должно указать этот флаг при первом отображении окна. |

Если окно перед этим было видимо, возвращается значение - не нуль. Если окно перед этим было не видимо, возвращаемое значение - нуль.

2) Функции для посыла сообщения окну.

* SendMessage

LRESULT SendMessage (HWND hWnd, UINT Msg, WPARAM wParam,

LPARAM lParam);

Функция SendMessage отправляет заданное сообщение окну или окнам. Функция вызывает оконную процедуру для заданного окна и не возвращает значение до тех пор, пока оконная процедура не обработает сообщение.

Параметры: hWnd - дескриптор окна, оконная процедура которого примет сообщение; Msg - определяет сообщение, которое будет отправлено; wParam, lParam - определяют дополнительную конкретизирующую сообщение информацию.

Величина возвращаемого значения определяет результат обработки сообщения; он зависит от отправленного сообщения.

* PostMessage

LRESULT PostMessage (HWND hWnd, UINT Msg, WPARAM wParam,

LPARAM lParam);

Функция PostMessage помещает (вставляет в очередь) сообщение в очередь сообщений, связанную с потоком, который создал заданное окно и возвращает значение без ожидания потока, который обрабатывает сообщение.

Параметры: hWnd - дескриптор окна, оконная процедура которого примет сообщение; Msg - определяет сообщение, которое будет поставлено в очередь; wParam, lParam - определяют дополнительную конкретизирующую сообщение информацию.

Если функция завершается успешно, величина возвращаемого значения - не нуль. Если функция завершается ошибкой, величина возвращаемого значения нулевая.

3) Поиск окон.

* FindWindow

HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWndName);

Функция FindWindow извлекает дескриптор окна верхнего уровня, имя класса и имя окна которого соответствуют заданным строкам. Эта функция не ищет дочерние окна. Эта функция не выполняет регистрозависимый поиск.

Параметры: lpClassName - указатель на строку с нулевым символом в конце, которая определяет имя класса или класс атома; lpWndName – указатель на строку с нулевым символом в конце, которая определяет имя окна (заголовок окна). Если этот параметр - ПУСТО (NULL), соответствуют имена всех окон.

Если функция завершилась успешно, возвращаемое значение - дескриптор окна, которое имеет заданное имя класса и имя окна. Если функция завершается ошибкой, возвращаемое значение - ПУСТО (NULL).

* GetWindow

HWND GetWindow (HWND hWnd, UINT uCmd);

Функция GetWindow извлекает дескриптор окна, который имеет определенное отношение к заданному окну.

Параметры: hWnd – дескриптор окна; uCmd – определяет отношение между заданным окном и окном, дескриптор которого должен быть извлечен. Этот параметр может быть одним из следующих значений:

|  |  |
| --- | --- |
| **Значение** | **Предназначение** |
| GW\_CHILD | Извлеченный дескриптор идентифицирует дочернее окно наверху Z - последовательности, если заданное окно - родительское окно; иначе, найденный дескриптор получит значение ПУСТО (NULL). Функция проверяет только дочерние окна заданного окна. |
| GW\_HWNDFIRST | Найденный дескриптор идентифицирует окно того же самого типа, которое является самым высоким в Z - последовательности. |
| GW\_HWNDLAST | Найденный дескриптор идентифицирует окно того же самого типа, который является самым низким в Z - последовательности. |
| GW\_HWNDNEXT | Найденный дескриптор идентифицирует окно ниже определяемого окна в Z - последовательности. |
| GW\_HWNDPREV | Найденный дескриптор идентифицирует окно выше определяемого окна в Z - последовательности. |
| GW\_OWNER | Найденный дескриптор идентифицирует окно владельца заданного окна, если таковой имеется. |

Если функция завершается успешно, возвращаемое значение - дескриптор окна. Если окно с заданным отношением к определяемому окну не существует, возвращаемое значение - ПУСТО (NULL).

* EnumChildWindows

BOOL EnumChildWindows (HWND hWndParent,

WNDENUMPROC lpEnumFunc, LPARAM lParam);

Функция EnumChildWindows перечисляет дочерние окна, которые принадлежат определенному родительскому окну, в свою очередь, передавая дескриптор каждого дочернего окна в функцию повторного вызова, определяемую программой. Функция работает до тех пор, пока не будет перечислено последнее дочернее окно или функция повторного вызова не возвратит значение ЛОЖЬ (FALSE).

Параметры: hWndParent - дескриптор родительского окна, дочерние окна которого должны быть перечислены; lpEnumFunc - указатель на определяемую программой функцию повторного вызова; lParam - устанавливает определяемое программой значение, которое будет передано в функцию повторного вызова.

Если функция завершилась успешно, возвращается значение не нуль. Если функция завершилась ошибкой, возвращаемое значение - ноль.

* EnumWindows

BOOL EnumWindows (WNDENUMPROC lpEnumFunc, LPARAM lParam);

Функция EnumWindows перечисляет все окна верхнего уровня на экране, передавая дескриптор каждого окна, в свою очередь, в определяемую программой функцию повторного вызова. EnumWindows действует до тех пор, пока последнее окно верхнего уровня не будет перечислено, или пока функция повторного вызова не возвратит значение ЛОЖЬ (FALSE).

Параметры: lpEnumFunc - указатель на определяемую программой функцию повторного вызова; lParam - устанавливает определяемое программой значение, которое будет передано в функцию повторного вызова.

Если функция завершилась успешно, возвращается значение не нуль. Если функция завершилась ошибкой, возвращаемое значение - ноль.

* FindWindowEx

HWND FindWindowEx (HWND hwndParent, HWND hwndChildAfter,

LPCTSTR lpszClass, LPCTSTR lpszWindow);

Функция FindWindowEx извлекает дескриптор окна, имя класса и имя окна которого соответствуют заданным строкам. Эта функция ищет дочерние окна, начиная с первого из последующих заданных дочерних окон. Эта функция не исполняет поиск, зависимый от регистра.

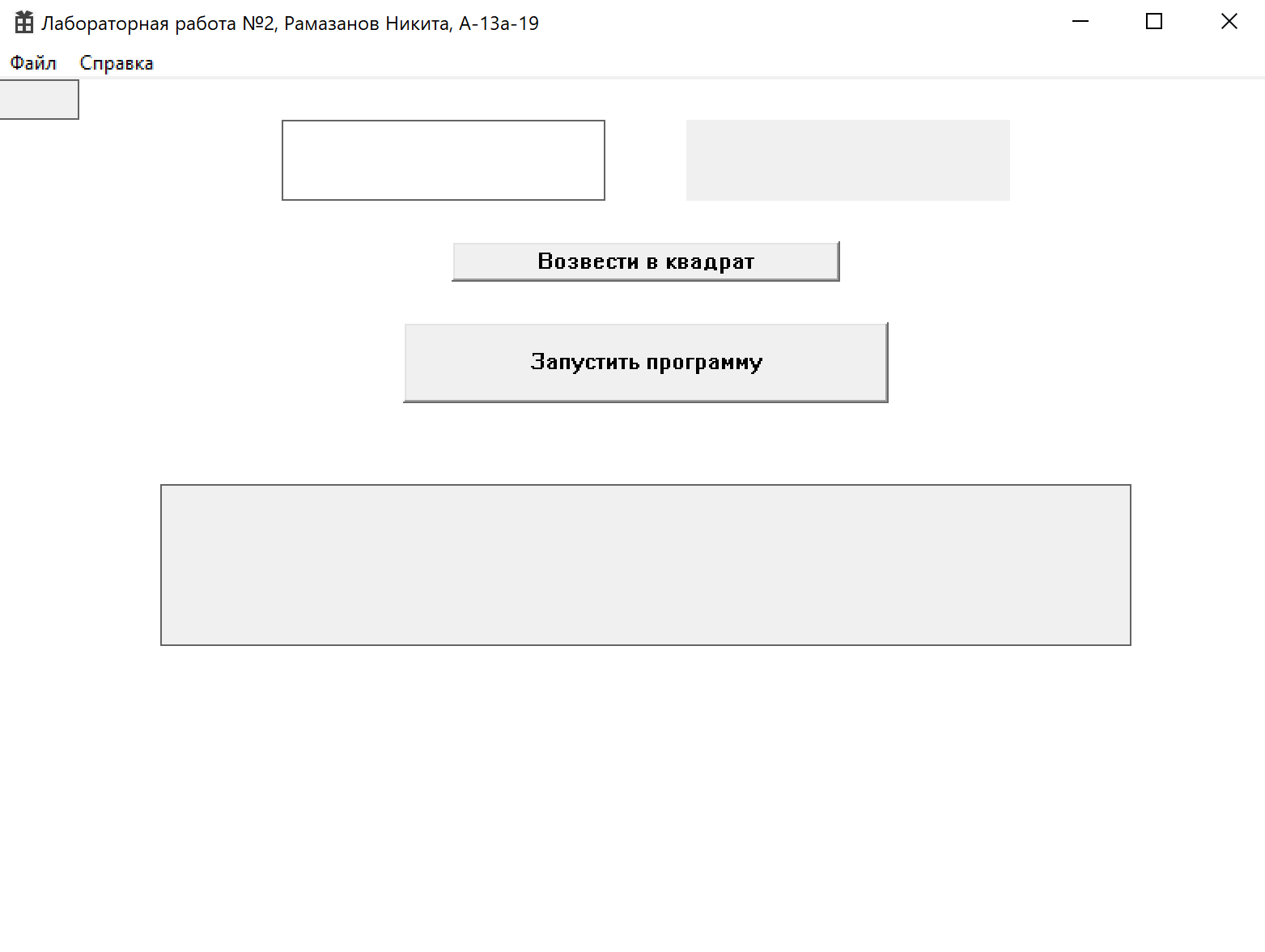
Параметры: hwndParent - дескриптор родительского окна, дочерние окна которого должны быть найдены; hwndChildAfter - дескриптор дочернего окна; lpszClass - указатель на строку с нулевым символом в конце, которая определяет имя класса или класс атома; lpszWindow - указывает на строку с нулевым символом в конце, которая определяет имя окна (заголовок окна). Если этот параметр - ПУСТО (NULL), соответствуют имена всех окон.

Если функция завершилась успешно, возвращаемое значение - дескриптор окна, которое имеет заданное имя класса и имя окна. Если функция завершается ошибкой, возвращаемое значение - ПУСТО (NULL).

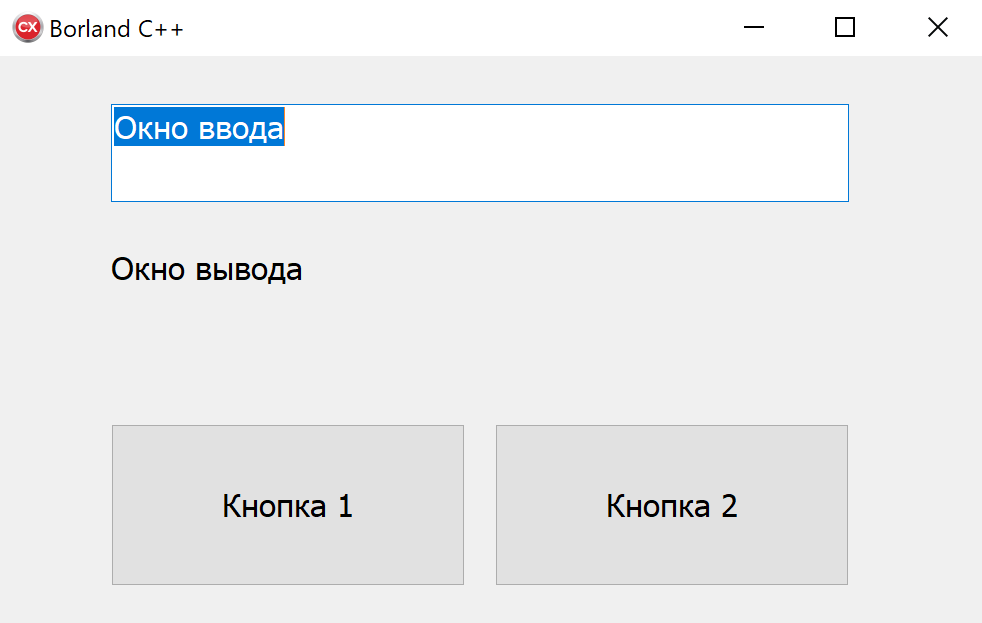
**2. Выполнение работы.**

1. Для программных экспериментов необходимы три вспомогательных приложения:

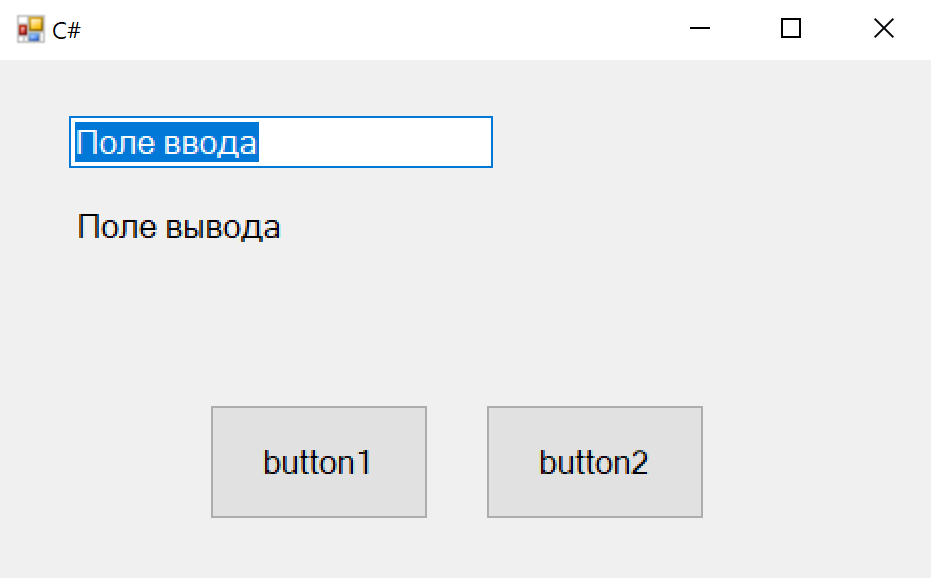
* + Программа из ЛР №2



* + Простейшая программа на Borland C++

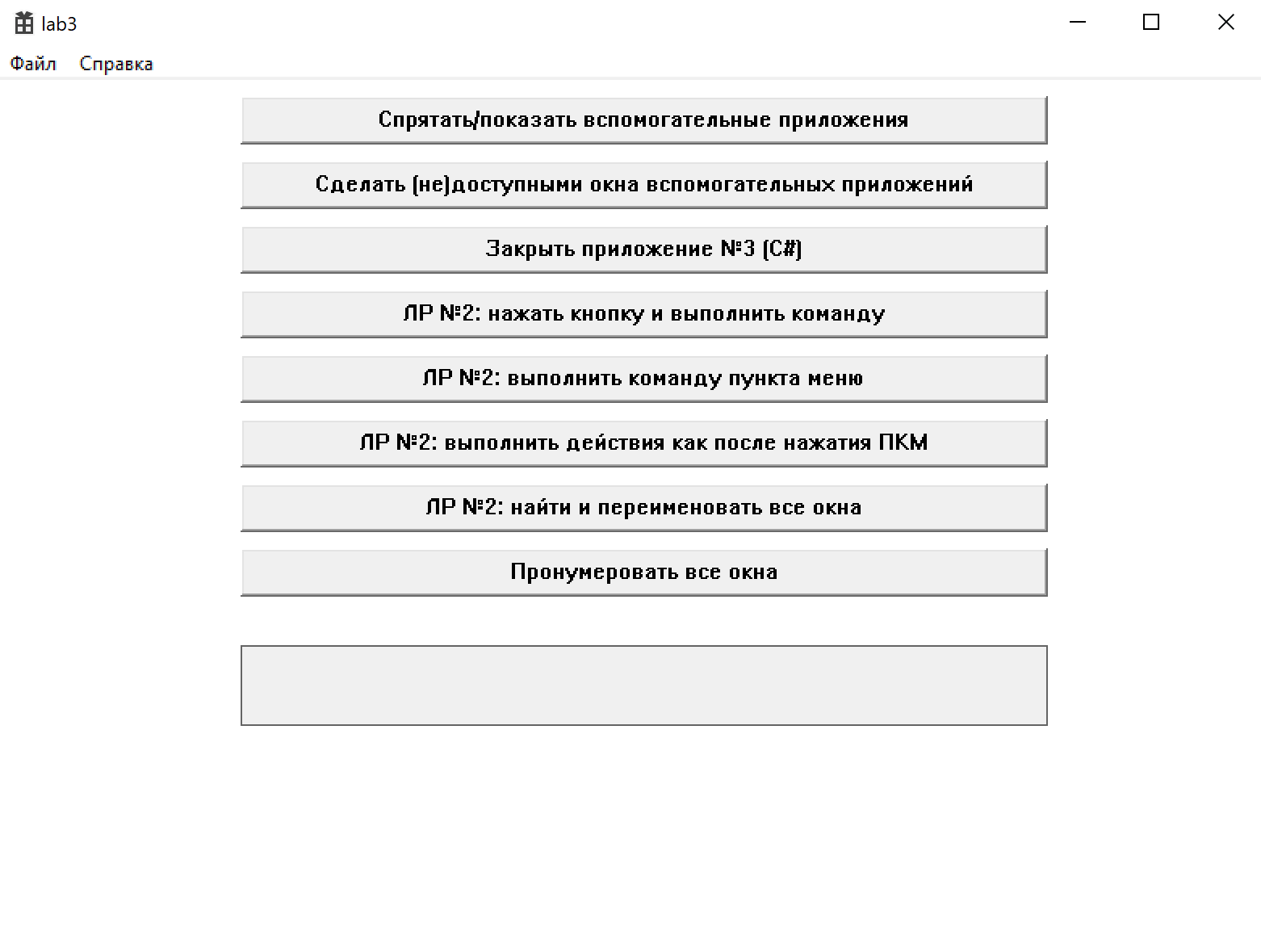


* + Простейшая программа на C#



2. Создал основное приложение Win32 Project (оно будет управлять окнами других приложений).

При создании окна ЛР №3 программа проверяет, созданы ли вспомогательные окна, и если нет, выводит соответствующее сообщение.



HWND BtnHideAuxApp; // Кнопка, которая прячет окно вспомогательного приложения

HWND BtnUnavailable; // Кнопка, делающая недоступным окно вспомогательного приложения

HWND BtnExitApp3; // Кнопка, закрывающая приложение на C#

HWND BtnPressBtn1Lab2; // Кнопка, нажимающая кнопку из ЛР №2

HWND BtnAboutLab2; // Кнопка, открывающая окно "О программе" из ЛР №2

HWND BtnRBDownLab2; // Кнопка-имитатор щелчка ПКМ из ЛР №2

HWND BtnMoveWndLab2; // Кнопка для перемещения окна из ЛР №2

HWND BtnRenameWndLab2; // Кнопка для переименования дочерних окон

HWND BtnNumerateWnd; // Кнопка для нумерации окон

HWND OutTxt; // Поле вывода

HWND lab2; // Окно ЛР №2

HWND Borland; // Окно программы Borland C++

HWND Sharp; // Окно программы на C#

// Идентификаторы кнопок и полей

const static int idBtnHideAuxApp = 1;

const static int idBtnUnavailable = 2;

const static int idBtnExitApp3 = 3;

const static int idBtnPressBtnLab2 = 4;

const static int idBtnAboutLab2 = 5;

const static int idBtnRBDownLab2 = 6;

const static int idBtnRenameWndLab2 = 7;

const static int idBtnNumerateWnd = 8;

const static int idOutTxt = 9;

int WWidth = 800;  
int WHeight = 600;

BtnHideAuxApp = CreateWindow(L"BUTTON", L"Спрятать/показать вспомогательные приложения", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 10, 500, 30, hWnd, (HMENU)idBtnHideAuxApp, hInstance, NULL);

BtnUnavailable = CreateWindow(L"BUTTON", L"Сделать (не)доступными окна вспомогательных приложений", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 50, 500, 30, hWnd, (HMENU)idBtnUnavailable, hInstance, NULL);

BtnExitApp3 = CreateWindow(L"BUTTON", L"Закрыть приложение №3 (C#)", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 90, 500, 30, hWnd, (HMENU)idBtnExitApp3, hInstance, NULL);

BtnPressBtn1Lab2 = CreateWindow(L"BUTTON", L"ЛР №2: нажать кнопку и выполнить команду", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 130, 500, 30, hWnd, (HMENU)idBtnPressBtnLab2, hInstance, NULL);

BtnAboutLab2 = CreateWindow(L"BUTTON", L"ЛР №2: выполнить команду пункта меню", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 170, 500, 30, hWnd, (HMENU)idBtnAboutLab2, hInstance, NULL);

BtnRBDownLab2 = CreateWindow(L"BUTTON", L"ЛР №2: выполнить действия как после нажатия ПКМ", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 210, 500, 30, hWnd, (HMENU)idBtnRBDownLab2, hInstance, NULL);

BtnRenameWndLab2 = CreateWindow(L"BUTTON", L"ЛР №2: найти и переименовать все дочерние окна", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 250, 500, 30, hWnd, (HMENU)idBtnRenameWndLab2, hInstance, NULL);

BtnNumerateWnd = CreateWindow(L"BUTTON", L"Пронумеровать все окна", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 150, 290, 500, 30, hWnd, (HMENU)idBtnNumerateWnd, hInstance, NULL);

OutTxt = CreateWindow(L"STATIC", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 150, 350, 500, 50, hWnd, (HMENU)idOutTxt, hInstance, NULL);

3. Добавил в основное приложение следующие возможности:

* + Спрятать окно вспомогательного приложения, если оно видимо;

case idBtnHideAuxApp:

IsWindowVisible(lab2) ? ShowWindow(lab2, SW\_HIDE) : ShowWindow(lab2, SW\_SHOW);

IsWindowVisible(Borland) ? ShowWindow(Borland, SW\_HIDE) : ShowWindow(Borland, SW\_SHOW);

IsWindowVisible(Sharp) ? ShowWindow(Sharp, SW\_HIDE) : ShowWindow(Sharp, SW\_SHOW);

break;

* + Сделать недоступным окно вспомогательного приложения;

case idBtnUnavailable:  
 IsWindowEnabled(lab2) ? EnableWindow(lab2, false) : EnableWindow(lab2, true);  
 IsWindowEnabled(Borland) ? EnableWindow(Borland, false) : EnableWindow(Borland, true);  
 IsWindowEnabled(Sharp) ? EnableWindow(Sharp, false) : EnableWindow(Sharp, true);  
break;

* + Закрыть приложение №3 (если оно работает и окно найдено).

case idBtnExitApp3:

SendMessage(Sharp, WM\_CLOSE, 0, 0);  
break;

4. Управление приложением из ЛР №2.

* + Нажать (программно) кнопку и выполнить команду;

case idBtnPressBtnLab2:

HWND hBtn;

hBtn = FindWindowEx(lab2, NULL, TEXT("BUTTON"), TEXT("Возвести в квадрат"));

if (hBtn)

SendMessage(hBtn, BM\_CLICK, 0, 0);

break;

* + Выполнить команду пункта меню;

case idBtnAboutLab2:

SendMessage(lab2, WM\_COMMAND, IDM\_ABOUT, 0);  
break;

* + Заставить программу выполнить действия, соответствующие нажатию пользователем правой кнопки мыши;

case idBtnRBDownLab2:

POINT pt;

GetCursorPos(&pt);

ScreenToClient(lab2, &pt);

SendMessage(lab2, WM\_RBUTTONDOWN, 0, MAKELPARAM(pt.x, pt.y));

break;

* + Заставить окно переместиться по экрану (послав, соответствующие сообщения, так как это окно умеет перемещаться при движении мыши в области клиента);

В данном пункте мне пришлось внести изменения в метод перемещения окна ЛР №2. Привожу их здесь:

BOOL moving;

int old\_x, old\_y;

case WM\_LBUTTONDOWN:

moving = TRUE;

old\_x = LOWORD(lParam);

old\_y = HIWORD(lParam);

break;

case WM\_LBUTTONUP:

moving = FALSE;

break;

case WM\_MOUSEMOVE:

if (moving) {

int new\_x = LOWORD(lParam), new\_y = HIWORD(lParam);

POINT old;

old.x = old\_x; old.y = old\_y;

POINT newcr;

newcr.x = new\_x; newcr.y = new\_y;

ClientToScreen(hWnd, &old);

ClientToScreen(hWnd, &newcr);

RECT rect;

GetWindowRect(hWnd, &rect);

MoveWindow(hWnd, rect.left + (newcr.x - old.x), rect.top + (newcr.y - old.y),

rect.right - rect.left, rect.bottom - rect.top, true);

ScreenToClient(hWnd, &newcr);

old\_x = newcr.x;

old\_y = newcr.y;

}

Теперь приведу код перемещения окна ЛР №2 через окно ЛР №3 (здесь мы просто передаем соответствующие сообщения окну второй лабораторной):

case WM\_LBUTTONDOWN:

{

SendMessage(lab2, WM\_LBUTTONDOWN, wParam, lParam);

}

break;

case WM\_MOUSEMOVE:

{

SendMessage(lab2, WM\_MOUSEMOVE, wParam, lParam);

}

break;

case WM\_LBUTTONUP:

{

SendMessage(lab2, WM\_LBUTTONUP, wParam, lParam);

}

break;

* + Найти и переименовать все дочерние окна вспомогательного приложения.

Код приведу ниже в 5-ом пункте (так как они схожи, решил их объединить).

5. В основное приложение добавить кнопку, при нажатии на которую программа найдет и пронумерует все запущенные в системе окна (а также и их дочерние). Окнам следует дать имена с номерами.

Инициализируем необходимые переменные:

int p = 0;

int param = 0;

int i = 0;

Объявим функцию обратного вызова для переименования и нумерации окон:

BOOL CALLBACK Numeration(HWND, LPARAM);

И определим ее:

BOOL CALLBACK Numeration(HWND tW, LPARAM lParam) {

if (param == 1) {

TCHAR lpWinTitle[256];

p++;

char z[10];

GetWindowTextW(tW, lpWinTitle, 256 - 1);

string str = "Окно №" + to\_string(p);

wstring s1 = wstring(str.begin(), str.end());

const wchar\_t\* s = s1.c\_str();

SendMessage(tW, WM\_SETTEXT, 0, (LPARAM)s);

TCHAR lpWinTitle1[256];

GetWindowText(tW, lpWinTitle1, 256 - 1);

param = 0;

i = 0;

EnumChildWindows(tW, &Numeration, 0);

param = 1;

return true;

}

else {

TCHAR lpWinTitle[256];

i++;

char z[10];

GetWindowText(tW, lpWinTitle, 256 - 1);

string str = "Дочернее окно №" + to\_string(i);

wstring s1 = wstring(str.begin(), str.end());

const wchar\_t\* s = s1.c\_str();

SendMessage(tW, WM\_SETTEXT, 0, (LPARAM)s);

TCHAR lpWinTitle1[256];

GetWindowText(tW, lpWinTitle1, 256 - 1);

return true;

}

}

Теперь кнопки:

case idBtnRenameWndLab2:

EnumChildWindows(lab2, &Numeration, 0);

break;

case idBtnNumerateWnd:

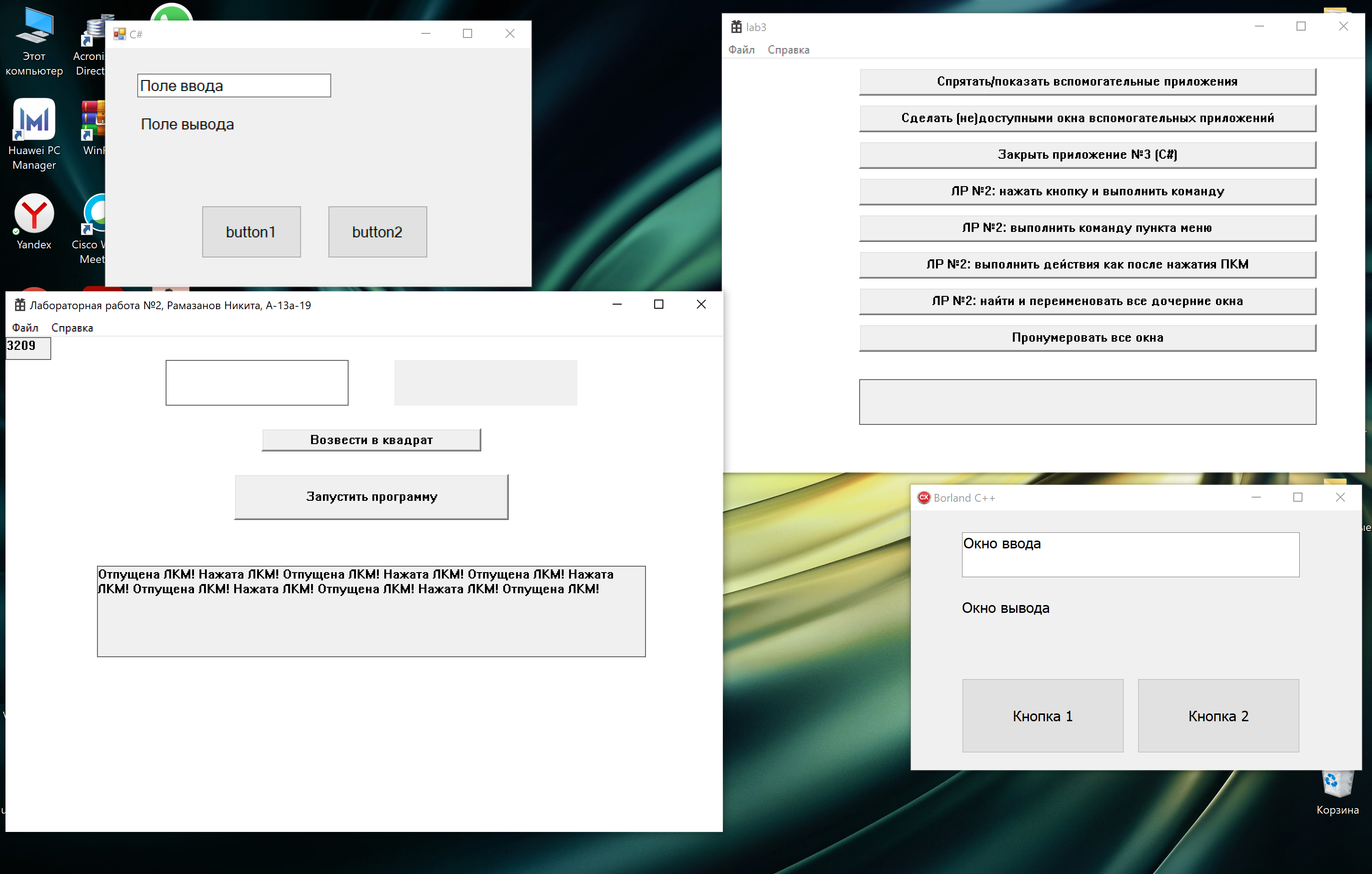
param = 1;

EnumWindows(&Numeration, 0);

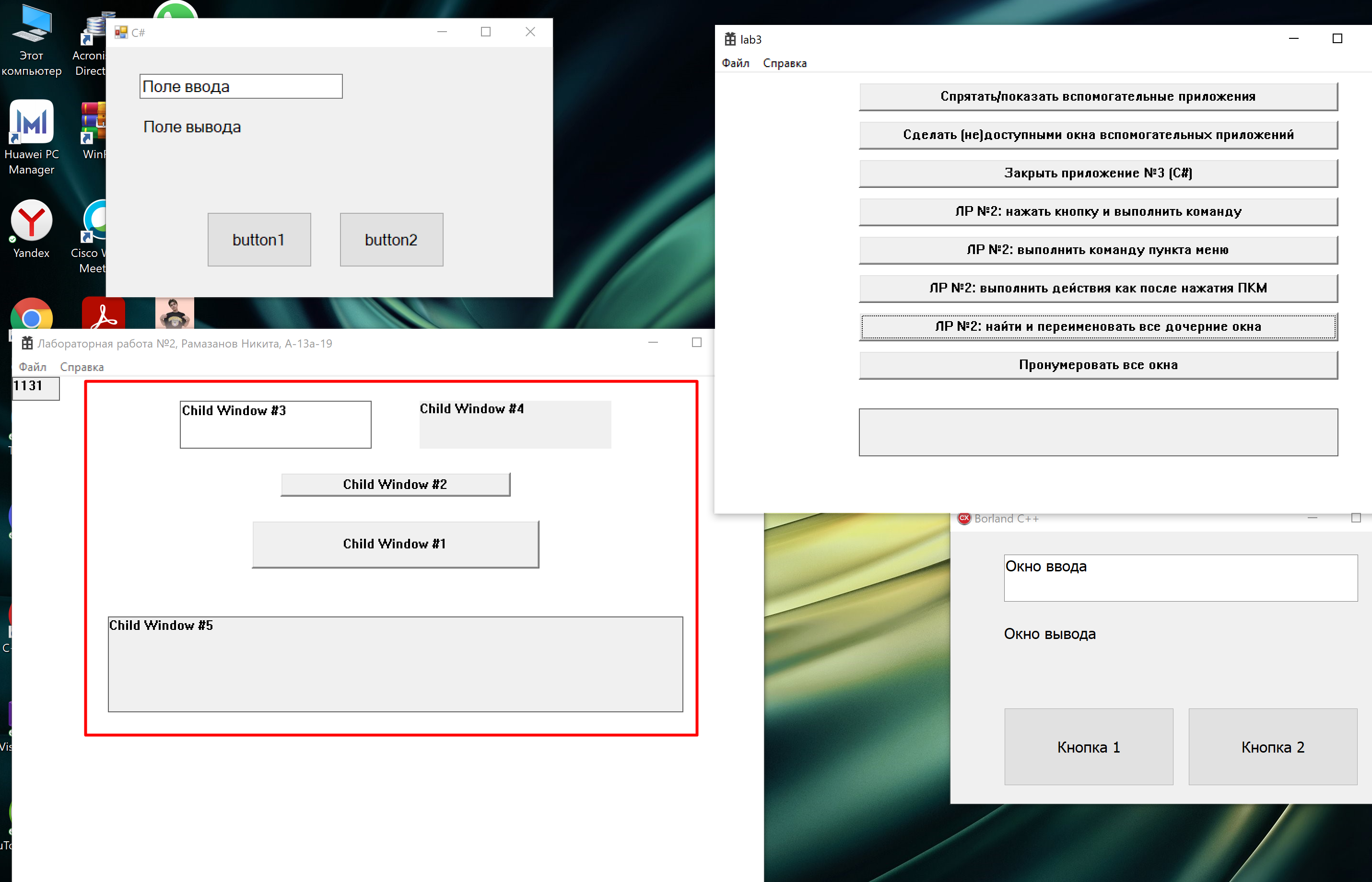
param = 0;

break;

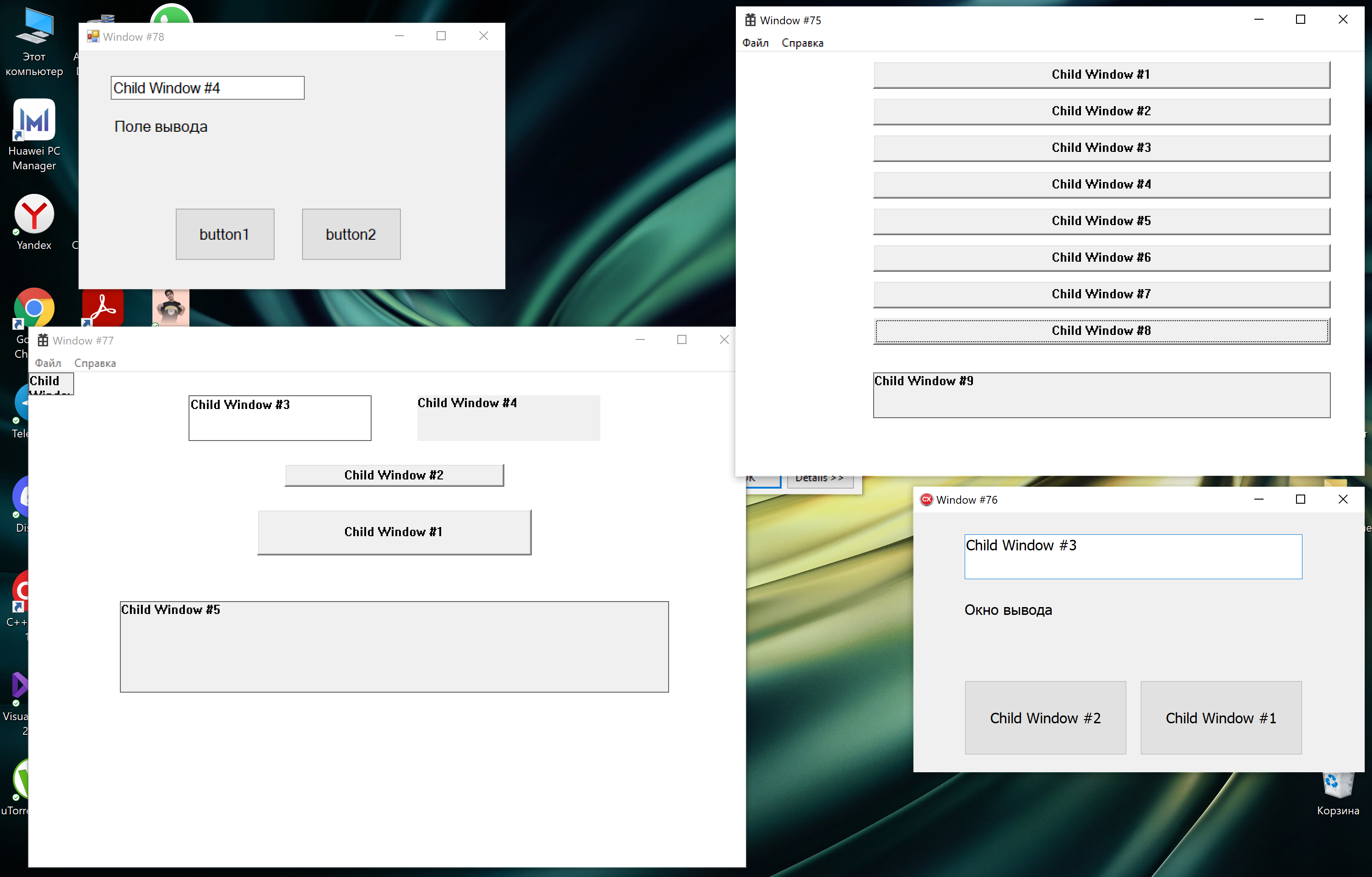
6. Запустить все приложения (четыре + плюс стандартные Windows и т.д. и пронумеровать окна (визуально определить примерное их количество, а также определить какие элементы являются окнами, а какие нет).

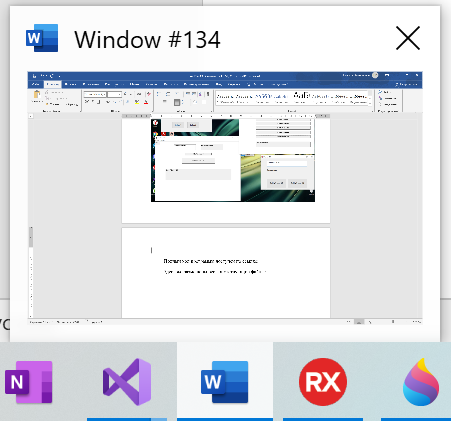
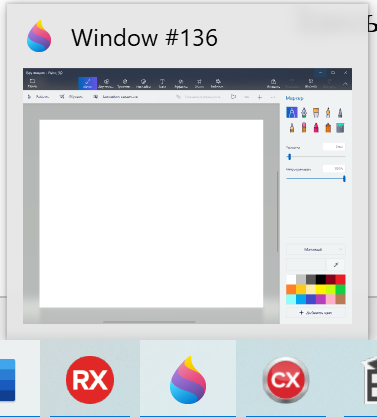


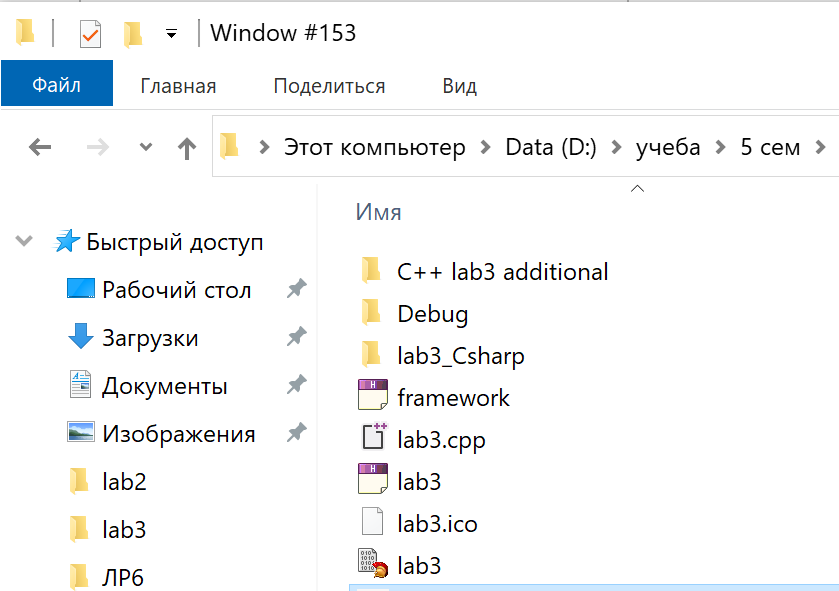
Сначала пронумеруем дочерние окна ЛР №2 (при этом оказалось, что и поля ввода, и поля вывода, и кнопки в Visual Studio на C++ считаются подчиненными окнами):



Теперь переименуем все окна в системе (оказалось, что в Visual Studio на C# и в Borland C++ поля вывода окнами не считаются):





Полный код программы доступен по ссылке: <https://github.com/programzan/SystemProgramming/blob/lab3/lab3.cpp>

Здесь же размещены все сопутствующие файлы: <https://github.com/programzan/SystemProgramming/tree/lab3>