МИНЕСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

Отчет к лабораторной работе № 3

Дисциплина: «Системное программирование»

Михаил Чув
[Дата]

Подготовка к лабораторной работе

Для выполнения лабораторной работы необходимо ознакомиться (по литературе или в сети) и сделать краткое описание следующих функций и параметров, а также используемых событий (также для работы будут необходимы функции и события, рассмотренные в работах № 1 и № 2):

1). Функции для работы с окнами

- **EnableWindow**

```
BOOL EnableWindow
(
  HWND hWnd,        // дескриптор окна
  BOOL bEnable      // флажок для включения или отключения ввода информации
);
```

Функция **EnableWindow** включает или отключает мышь и ввод с клавиатуры в определенном окне или элементе управления. Когда ввод заблокирован, окно не принимает ввод типа щелчков мыши и нажатий клавиш. Когда ввод включен, окно принимает всю вводимую информацию. Если окно было предварительно заблокировано, возвращаемое значение не ноль. Если окно предварительно не было заблокировано, возвращаемое значение нулевое.

- **IsWindowVisible**

```
BOOL IsWindowVisible
(
HWND hWnd         // дескриптор окна
);
```

Функция **IsWindowVisible** находит данные о состоянии видимости заданного окна. Если определяемое окно и его родительское окно имеют стиль **WS_VISIBLE**, возвращаемое значение отлично от нуля. Если определяемое окно и его родительское окно имеют стиль **WS_VISIBLE**, возвращаемое значение отлично от нуля.

- **IsWindowEnabled**

```
BOOL IsWindowEnabled
(
        HWND hWnd         // дескриптор окна
);
```

Функция **IsWindowEnabled** устанавливает, включено ли заданное окно для ввода информации от мыши и клавиатуры. Если окно включено, величина возвращаемого значения отличная от нуля.

- **ShowWindow**

```
BOOL ShowWindow
(
HWND hWnd,        // дескриптор окна
int nCmdShow      // состояние показа окна
);
```

Функция **ShowWindow** устанавливает состояние показа определяемого окна. Если функция завершилась успешно, возвращается значение отличное от нуля.

Значения для nCmdShow:

- SW_HIDE - Скрывает окно и активизирует другое окно.
- SW_MAXIMIZE - Развертывает определяемое окно.
- SW_MINIMIZE - Свертывает определяемое окно и активизирует следующее окно верхнего уровня в Z-последовательности.

- SW_RESTORE - Активизирует и отображает окно. Если окно свернуто или развернуто, Windows восстанавливает в его первоначальных размерах и позиции. Прикладная программа должна установить этот флажок при восстановлении свернутого окна.
- SW_SHOW - Активизирует окно и отображает его текущие размеры и позицию.
- SW_SHOWDEFAULT - Устанавливает состояние показа, основанное на флажке SW_, определенном в структуре STARTUPINFO, переданной в функцию CreateProcess программой, которая запустила прикладную программу.
- SW_SHOWMAXIMIZED - Активизирует окно и отображает его как развернутое окно.
- SW_SHOWMINIMIZED - Активизирует окно и отображает его как свернутое окно.
- SW_SHOWMINNOACTIVE - Отображает окно как свернутое окно. Активное окно остается активным.
- SW_SHOWNA - Отображает окно в его текущем состоянии. Активное окно остается активным.
- SW_SHOWNOACTIVATE - Отображает окно в его самом современном размере и позиции. Активное окно остается активным.
- SW_SHOWNORMAL - Активизирует и отображает окно. Если окно свернуто или развернуто, Windows восстанавливает его в первоначальном размере и позиции. Прикладная программа должна установить этот флажок при отображении окна впервые.

2). Послать сообщение окну:

- **SendMessage**

```
LRESULT SendMessage(
  HWND hWnd, // дескриптор окна, принимающего сообщение
  UINT Msg, // определяет сообщение, которое будет отправлено.
  WPARAM wParam, // доп. информация
  LPARAM lParam // доп. информация
);
```

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.

Чтобы отправить сообщение и возвратить немедленно значение, используйте функцию SendMessageCallback или SendNotifyMessage. Чтобы поместить сообщение в очередь сообщений потока и возвратить немедленно значение, используйте функцию PostMessage или PostThreadMessage.

Если параметр hWnd имеет значение HWND_BROADCAST, сообщение отправляется всем окнам верхнего уровня в системе, включая заблокированные или невидимые, не имеющие владельца, перекрывающие и выскакивающие окна; но сообщение не отправляется дочерним окнам.
Если функция завершается успешно, величина возвращаемого значения - не нуль.

- **PostMessage**

```
BOOL PostMessage(
        HWND hWnd,  // Дескриптор окна, оконная процедура которого должна принять сообщение.
        UINT Msg,   // Определяет сообщение, которое должно быть поставлено в очередь.
        WPARAM wParam,      // Доп. информация
        LPARAM lParam       // Доп. Информация
);
```

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

Например,

Выполнить пункт меню:

```
SendMessage(handleW1,WM_COMMAND,MAKELONG(32771,0),0);
```

Изменить заголовок окна:

```
SendMessage(handleW1,WM_SETTEXT,0,LPARAM(LPCTSTR("I See you")));
```

Послать сообщение о нажатии мыши:

```
SendMessage(hWnd,WM_LBUTTONDOWN,MK_LBUTTON,MAKELONG(100,100));
```

3). Поиск окон:

- FindWindow

```
HWND FindWindow
(
LPCTSTR lpClassName,     // указатель на имя класса
LPCTSTR lpWindowName     // указатель на имя окна
);
```

Функция **FindWindow** разыскивает данные о дескрипторе окна верхнего уровня, чье имя класса и имя окна соответствуют определенным строкам. Эта функция не ищет дочерние окна. Если функция завершилась успешно, возвращаемое значение - дескриптор окна, которое имеет определенное имя класса и имя окна, иначе - NULL.

Пример: `handleWindow = FindWindow(0,"Мое окно");`

- GetWindow

```
HWND GetWindow
(
HWND hWnd,               // дескриптор первоначального окна
UINT uCmd                // флажок отношения
);
```

Функция **GetWindow** отыскивает дескриптор окна, который имеет определенное отношение (**Z** - последовательность или владелец) к заданному окну. Если функция завершается успешно, возвращаемое значение - дескриптор окна, иначе - NULL.

Пример: `handleW2 = GetWindow(handleW1,GW_CHILD);`

- EnumChildWindows

```
BOOL EnumChildWindows
(
HWND hWndParent,             // дескриптор родительского окна
WNDENUMPROC lpEnumFunc,      // указатель на функцию обратного вызова
LPARAM lParam                // значение, определяемое программой
);
```

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

Пример: EnumChildWindows(handleW,&EnumCW,0); (функция обратного вызова должна быть реализована в программном коде и ее объявление добавлено в начало файла: BOOL CALLBACK EnumCW(HWND, LPARAM);)

- EnumWindows

```
BOOL EnumWindows
(
WNDENUMPROC lpEnumFunc,    // указатель на функцию обратного вызова
LPARAM lParam              // определяемое программой значение
);
```

Функция **EnumWindows** перечисляет все окна верхнего уровня на экране, передавая дескриптор каждого окна, в свою очередь, в определяемую программой функцию обратного вызова. **EnumWindows** действует до тех пор, пока последнее окно верхнего уровня не будет перечислено, или пока функция обратного вызова не возвратит значение ЛОЖЬ (**FALSE**). Если функция завершилась успешно, возвращается значение отличное от нуля.

- **FindWindowEx**

```
HWND FindWindowEx
(
HWND hwndParent,                // дескриптор родительского окна
HWND hwndChildAfter,            // дескриптор дочернего окна
LPCTSTR lpszClass,              // указатель имени класса
LPCTSTR lpszWindow              // указатель имени окна
);
```

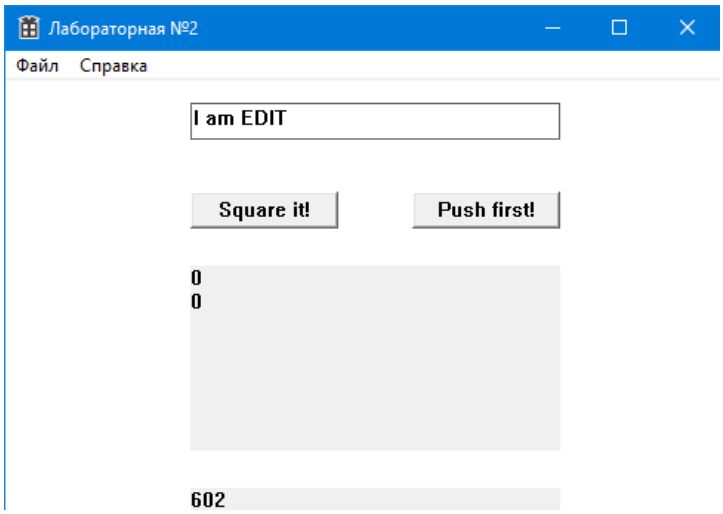
Функция **FindWindowEx** отыскивает данные о дескрипторе окна, имя класса и имя окна которого соответствуют определенным строкам. Функция поиска дочерних окон начинается с первого до последнего заданного дочернего окна. Если функция завершается успешно, возвращаемое значение - дескриптор окна, которое имеет определенный класс и имена окон, иначе - NULL.

Например, `hBut = FindWindowEx(handleW,0,"BUTTON",NULL);`

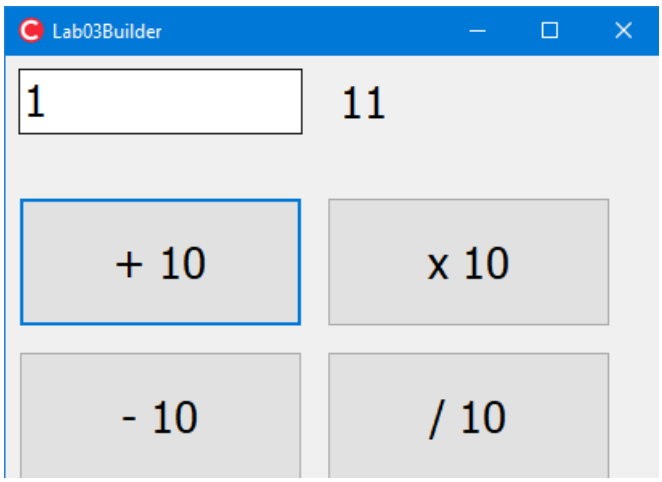
Задание

1. Для программных экспериментов необходимы три вспомогательных приложения:

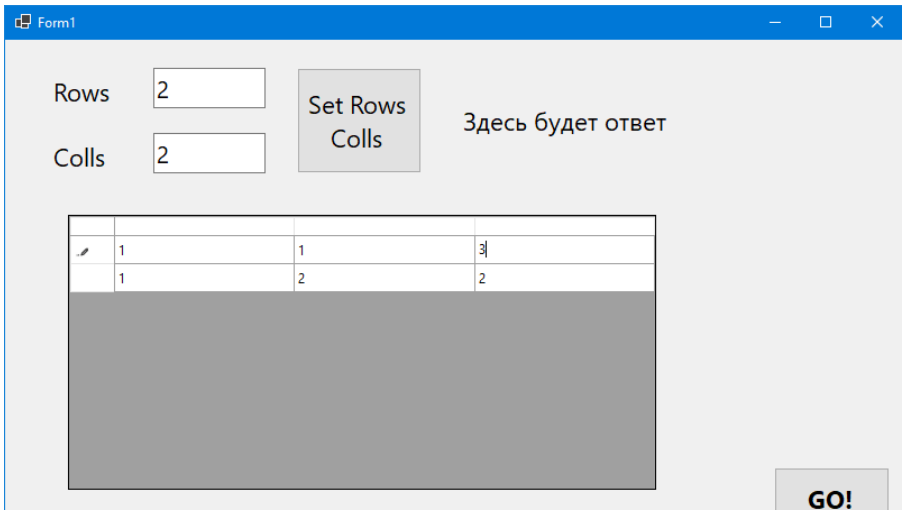
   - Первое: программа из лаб. работы № 2



   - Второе: простейшая программа на Borland C++ (окно, поля ввода и вывода и две кнопки с обработчиками событий, можно также добавить другие элементы управления)



- Третье: простейшая программа на C# (также с кнопками и разнообразными элементами управления)



2. Создать основное приложение Win32 Project (оно будет управлять окнами других приложений)
3. Добавить в основное приложение следующие возможности:

   - Спрятать окно вспомогательного приложения, если оно видимо;

Добавим область глобальных переменных дескрипторы кнопок с идентификаторами:

```
// Дескрипторы для кнопок сокрытия окон
HWND btnHideLab02;
const int idBtnHideLab02 = 0;
HWND btnHideCppBuilder;
const int idBtnHideCppBuilder = 1;
HWND btnHideCSharp;
const int idBtnHideCSharp = 2;
```

Имена вспомогательных приложений:

```
// Имена окон
LPCWSTR lab02Name = L"Лабораторная №2";
LPCWSTR cppBuilderName = L"Lab03Builder";
LPCWSTR cSharpName = L"Form1";
```

А также дескрипторы вспомогательных окон:

```
// Дескрипторы для управляемых окон
```

```
	HWND hLab02;
	HWND hCppBuilder;
	HWND hCSharp;
```

Создадим три кнопки на сокрытие окон (лабораторная №2, приложение C++Builder приложение на C# соответственно):

```
	btnHideLab02 = CreateWindow(L"BUTTON", L"Hide/Show Lab02", WS_CHILD | 
BS_PUSHBUTTON | WS_VISIBLE, xoff, yRow, gridWidth, gridHeight, hWnd, 
(HMENU)idBtnHideLab02, hInstance, 0);

	btnHideCppBuilder = CreateWindow(L"BUTTON", L"Hide/Show Builder", WS_CHILD | 
BS_PUSHBUTTON | WS_VISIBLE, gridWidth + xoff * 2, yRow, gridWidth, gridHeight, hWnd, 
(HMENU)idBtnHideCppBuilder, hInstance, 0);

	btnHideCSharp = CreateWindow(L"BUTTON", L"Hide/Show C#", WS_CHILD | BS_PUSHBUTTON 
| WS_VISIBLE, gridWidth * 2 + xoff * 3, yRow, gridWidth, gridHeight, hWnd, 
(HMENU)idBtnHideCSharp, hInstance, 0);
```

И напишем обработчики нажатий на эти кнопки (используем функции `FindWindow()`, `IsWindowVisible()`, `ShowWindow()`):

```
case idBtnHideLab02:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (IsWindowVisible(hLab02))
		ShowWindow(hLab02, SW_HIDE);
	else
		ShowWindow(hLab02, SW_SHOWNOACTIVATE);
}
break;
case idBtnHideCppBuilder:
{
	hCppBuilder = FindWindow(NULL, cppBuilderName);
	if (IsWindowVisible(hCppBuilder))
		ShowWindow(hCppBuilder, SW_HIDE);
	else
		ShowWindow(hCppBuilder, SW_SHOWNOACTIVATE);
}
break;
case idBtnHideCSharp:
{
	hCSharp = FindWindow(NULL, cSharpName);
	if (IsWindowVisible(hCSharp))
		ShowWindow(hCSharp, SW_HIDE);
	else
		ShowWindow(hCSharp, SW_SHOWNOACTIVATE);
}
break;
```

- Сделать недоступным окно вспомогательного приложения;

Добавим еще три кнопки:

```
// Дескрипторы для кнопок переключения доступности окон
HWND btnDisableLab02;
const int idBtnDisableLab02 = 3;
HWND btnDisableCppBuilder;
const int idBtnDisableCppBuilder = 4;
HWND btnDisableCSharp;
const int idBtnDisableCSharp = 5;

	btnPushSquareItLab02 = CreateWindow(L"BUTTON", L"Push Square It", WS_CHILD | 
BS_PUSHBUTTON | WS_VISIBLE,  xoff, yRow, gridWidth, gridHeight, hWnd, 
(HMENU)idBtnPushSquareItLab02, hInstance, 0);
```

```
    btnRenameChlidrenCppBuilder = CreateWindow(L"BUTTON", L"Rename Chidren", WS_CHILD |
BS_PUSHBUTTON | WS_VISIBLE,  gridWidth + xoff * 2, yRow, gridWidth, gridHeight, hWnd,
(HMENU)idBtnRenameChlidrenCppBuilder, hInstance, 0);

    btnCloseCSharp = CreateWindow(L"BUTTON", L"Close C#", WS_CHILD | BS_PUSHBUTTON |
WS_VISIBLE,  gridWidth * 2 + xoff * 3, yRow, gridWidth, gridHeight, hWnd,
(HMENU)idBtnCloseCSharp, hInstance, 0);
```

Напишем обработчики нажатий на эти кнопки (используем функции `FindWindow()`, `IsWindowEnabled()`, `EnableWindow()`):

```
case idBtnDisableLab02:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (IsWindowEnabled(hLab02))
		EnableWindow(hLab02, FALSE);
	else
		EnableWindow(hLab02, TRUE);
}
break;
case idBtnDisableCppBuilder:
{
	hCppBuilder = FindWindow(NULL, cppBuilderName);
	if (IsWindowEnabled(hCppBuilder))
		EnableWindow(hCppBuilder, FALSE);
	else
		EnableWindow(hCppBuilder, TRUE);
}
break;
case idBtnDisableCSharp:
{
	hCSharp = FindWindow(NULL, cSharpName);
	if (IsWindowEnabled(hCSharp))
		EnableWindow(hCSharp, FALSE);
	else
		EnableWindow(hCSharp, TRUE);
}
break;
```

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

Добавим еще одну кнопку:

```
// Кнопка закрытия окна 3 (приложение на C#)
HWND btnCloseCSharp;
const int idBtnCloseCSharp = 6;

btnCloseCSharp = CreateWindow(L"BUTTON", L"Close C#", WS_CHILD | BS_PUSHBUTTON |
WS_VISIBLE,  gridWidth * 2 + xoff * 3, yRow, gridWidth, gridHeight, hWnd,
(HMENU)idBtnCloseCSharp, hInstance, 0);
```

Напишем обработчик нажатия на эту кнопку (используем функции `FindWindow()`, `IsWindowEnabled()`, `PostMessage()`):

```
case idBtnCloseCSharp:
{
	hCSharp = FindWindow(NULL, cSharpName);
	if (hCSharp && IsWindowEnabled(hCSharp))
		PostMessage(hCSharp, WM_QUIT, NULL, NULL);
}
break;
```

4. Управление приложением из лаб. работы № 2:

- Нажать (программно) кнопку и выполнить команду;

Добавим еще одну кнопку, а также кнопку для следующего пункта задания:

```
// Кнопка для программного нажатия кнопки
// И выполнения команды пункта меню (вызов окна About)
HWND btnPushSquareItLab02;
const int idBtnPushSquareItLab02 = 7;
HWND btnSelectAboutMenuItemLab02;
const int idBtnSelectAboutMenuItemLab02 = 8;

btnPushSquareItLab02 = CreateWindow(L"BUTTON", L"Push Square It", WS_CHILD |
BS_PUSHBUTTON | WS_VISIBLE,  xoff, yRow, gridWidth, gridHeight, hWnd,
(HMENU)idBtnPushSquareItLab02, hInstance, 0);
```

Так как нужно изменить найти конкретный дочерний элемент, в обработчике воспользуемся функцией `FindWindowEx()`:

```
case idBtnPushSquareItLab02:
{
	hLab02 = FindWindow(NULL, lab02Name);
	HWND hBtnSquareIt = NULL;
	if (hLab02)
		hBtnSquareIt = FindWindowEx(hLab02, NULL, L"BUTTON", L"Square it!");
	if (hBtnSquareIt)
		SendMessage(hBtnSquareIt, BM_CLICK, 0, 0);
}
break;
```

- Выполнить команду пункта меню;

Так как в Лабораторной работе №2 были только автоматически созданные пункты меню, будем вызывать пункт меню, открывающий окно «О программе»:

```
	btnSelectAboutMenuItemLab02 = CreateWindow(L"BUTTON", L"About Lab02", WS_CHILD |
BS_PUSHBUTTON | WS_VISIBLE,  xoff, yRow, gridWidth, gridHeight, hWnd,
(HMENU)idBtnSelectAboutMenuItemLab02, hInstance, 0);

case idBtnSelectAboutMenuItemLab02:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (hLab02)
		SendMessage(hLab02, WM_COMMAND, IDM_ABOUT, NULL);
	}
break;
```

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

Так как в Лабораторной работе №2 при нажатии на правую кнопку выводились клиентские координаты, будем посылать нажатие ПКМ:

```
case WM_RBUTTONDOWN:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (hLab02)
	SendMessage(hLab02, WM_RBUTTONDOWN, MK_RBUTTON, lParam);
	}
break;
```

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

При посылке сообщения о передвижении мыши необходимо провести преобразование координат:

```
case WM_LBUTTONDOWN:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (hLab02)
		SendMessage(hLab02, WM_LBUTTONDOWN, MK_LBUTTON, lParam);
}
break;
case WM_LBUTTONUP:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (hLab02)
		SendMessage(hLab02, WM_LBUTTONUP, MK_LBUTTON, lParam);
}
break;
case WM_MOUSEMOVE:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (hLab02) {
		POINT coords;
		coords.x = LOWORD(lParam);
		coords.y = HIWORD(lParam);
		ClientToScreen(hWnd, &coords);
		ScreenToClient(hLab02, &coords);
		SendMessage(hLab02, WM_MOUSEMOVE, wParam, MAKELONG(coords.x, coords.y));
	}
}
break;
```

- Найти и переименовать все дочерние окна вспомогательного приложения.

Создадим еще одну кнопку для этого (п):

```
// Кнопка переименования дочерних окон
HWND btnRenameChlidrenLab02;
const int idBtnRenameChlidrenLab02 = 9;

btnRenameChlidrenLab02 = CreateWindow(L"BUTTON", L"Rename Chidren", WS_CHILD |
BS_PUSHBUTTON | WS_VISIBLE,  xoff, yRow, gridWidth, gridHeight, hWnd,
(HMENU)idBtnRenameChlidrenLab02, hInstance, 0);

case idBtnRenameChlidrenLab02:
{
	hLab02 = FindWindow(NULL, lab02Name);
	if (hLab02)
		EnumChildWindows(hLab02, &RenameChildren, LPARAM(L"Renamed!"));
}
break;
```

Так как функция `EnumChildWindows()` принимает указатель на функцию обратного вызова, напишем эту функцию:

```
BOOL CALLBACK RenameChildren(HWND hWnd, LPARAM lParam){
    SendMessage(hWnd, WM_SETTEXT, 0, lParam);
    return TRUE;
}
```

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 (например, Окно1, Окно2, Дочернее3, Дочернее4 и т.д.).

```
// Кнопка подсчета и переименования всех окон
HWND btnCountAndRename;
const int idBtnCountAndRename = 10;


btnCountAndRename = CreateWindow(L"BUTTON", L"Count and Rename", WS_CHILD | BS_PUSHBUTTON
| WS_VISIBLE,  xoff, yRow, gridWidth * 3 + 2 * xoff, gridHeight, hWnd,
(HMENU)idBtnCountAndRename, hInstance, 0);


case idBtnCountAndRename:
{
       windowCounter = 0;
       EnumWindows(&RenameWindow, NULL);
}
break;
```

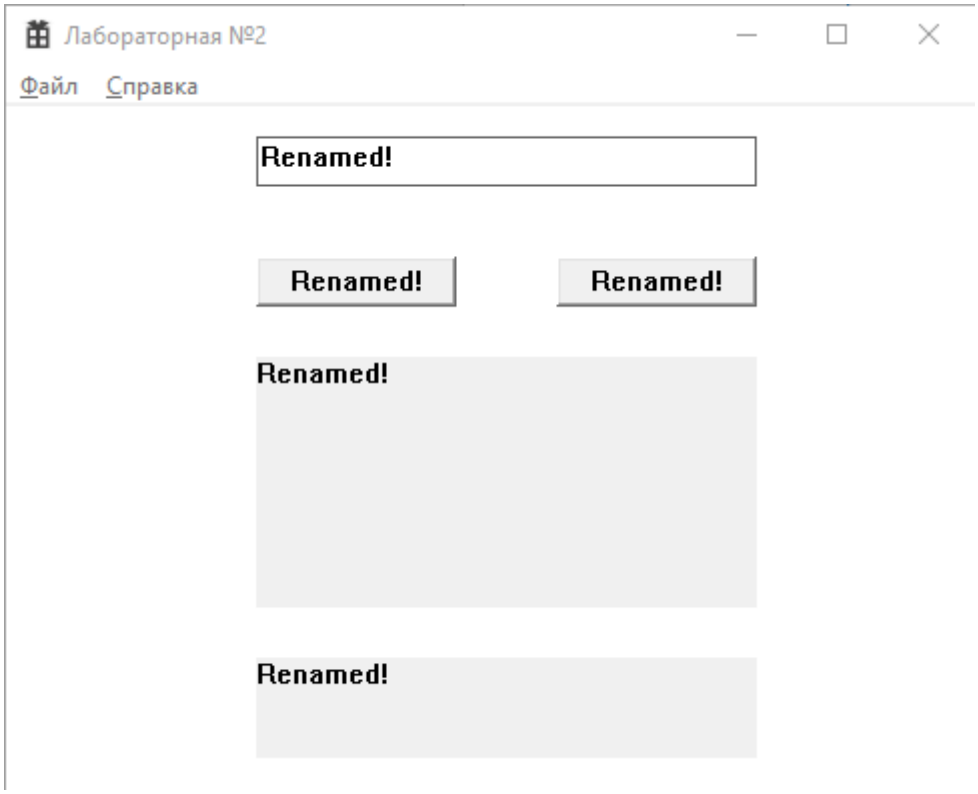
Так как функция `EnumWindows()` принимает указатель на функцию обратного вызова, напишем эту функцию, в ней будем переименовывать окно с учетом номера, а также пробегать по дочерним окнам с помощью функции `EnumChildWindows()`:

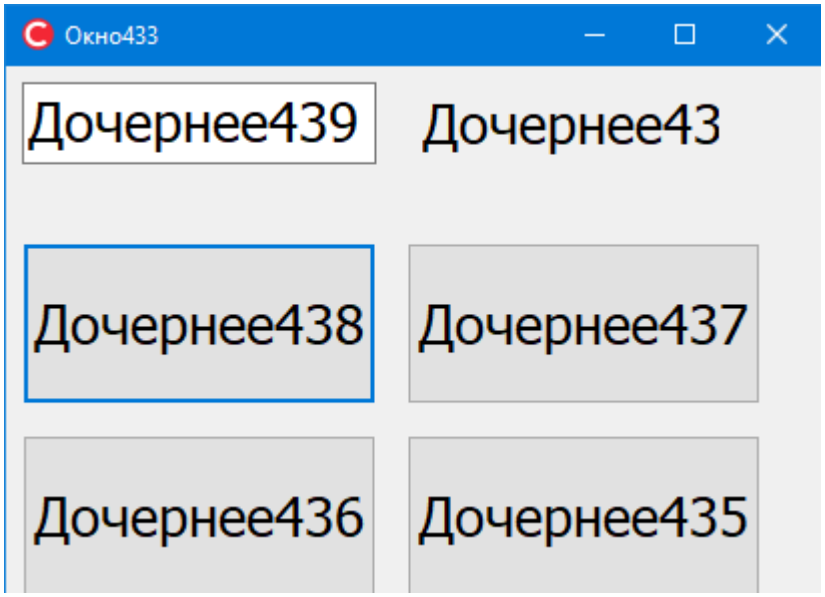
```
BOOL CALLBACK RenameWindow(HWND hWnd, LPARAM lParam) {
    windowCounter++;
    TCHAR newName[MAX_LOADSTRING] = _T("Окно");
    TCHAR windowNumber[10];
    _itot_s(windowCounter, windowNumber, 10);
    _tcscat_s(newName, windowNumber);
    SendMessage(hWnd, WM_SETTEXT, NULL, LPARAM(newName));
    EnumChildWindows(hWnd, &RenameChild, lParam);
    return TRUE;
}
```

Функция `EnumChildWindows()` также принимает указатель на функцию обратного вызова:

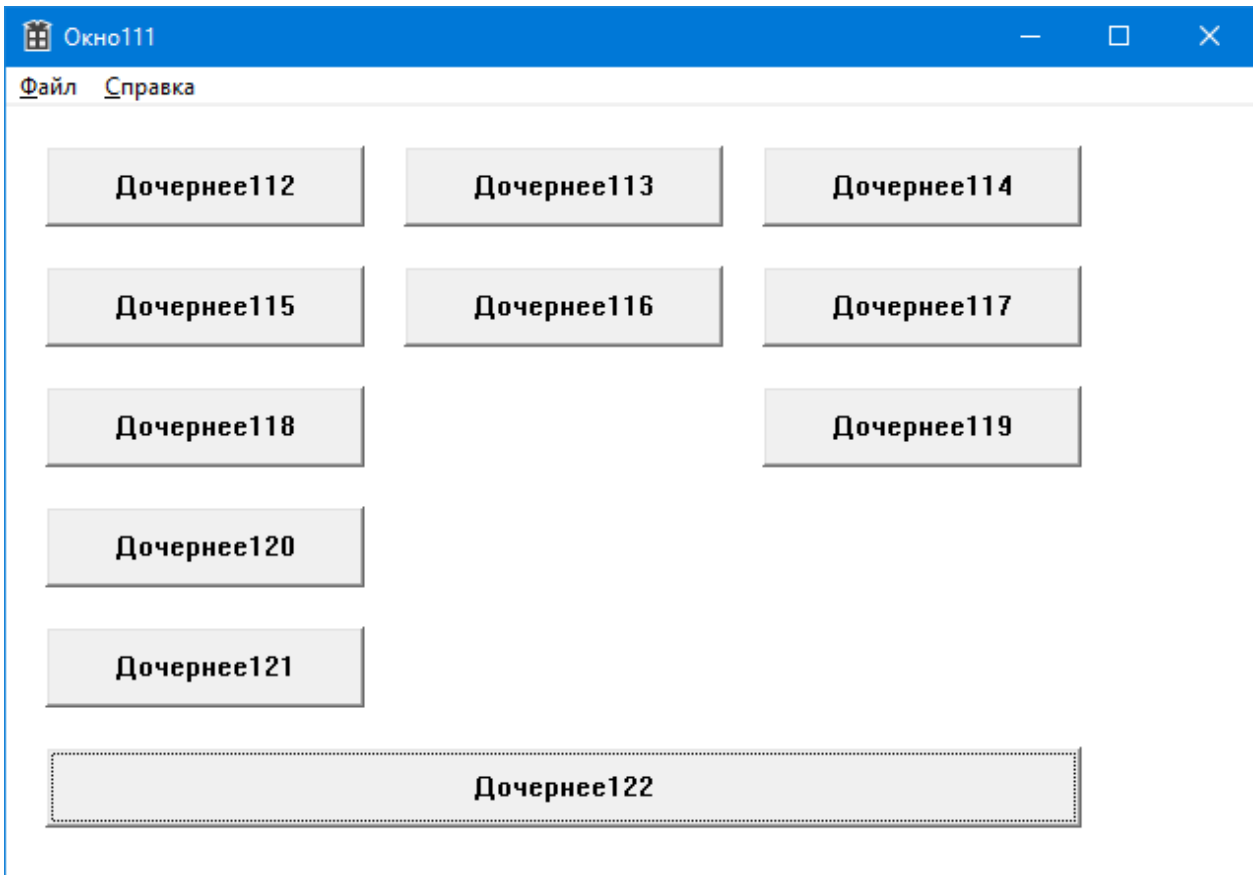
```
BOOL CALLBACK RenameChild(HWND hWnd, LPARAM lParam) {
    windowCounter++;
    TCHAR newName[MAX_LOADSTRING] = _T("Дочернее");
    TCHAR windowNumber[10];
    _itot_s(windowCounter, windowNumber, 10);
    _tcscat_s(newName, windowNumber);
    SendMessage(hWnd, WM_SETTEXT, NULL, LPARAM(newName));
    return TRUE;
}
```

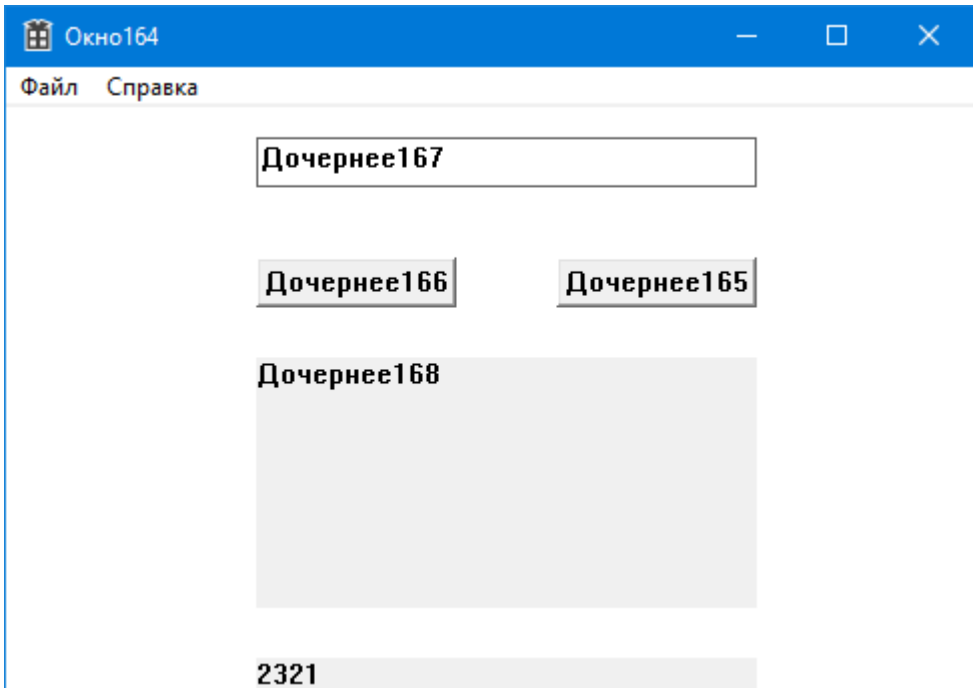
6. Запустить все приложения (четыре + плюс стандартные Windows и т.д. и пронумеровать окна (визуально определить примерное их количество, а также определить какие элементы являются окнами, а какие нет).



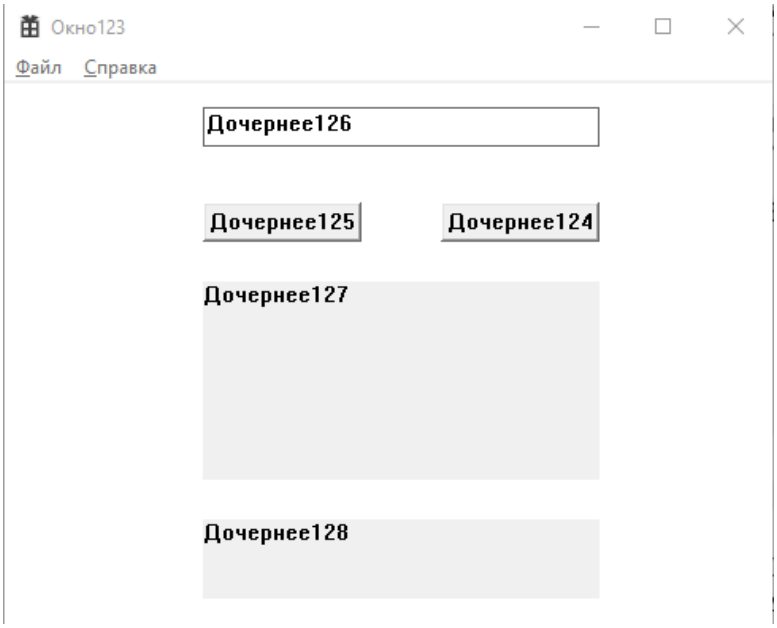


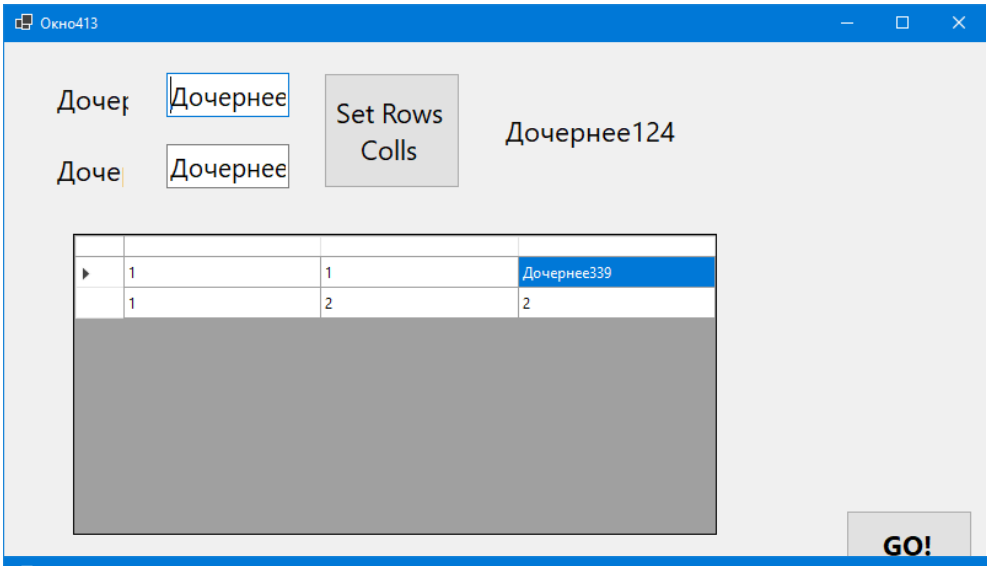
(Здесь в качестве элемента для вывода был выбран TStaticText, который реагирует на сообщение WM_SETTEXT, если же выбрать элемент TLabel, то он свое содержимое менять не будет)

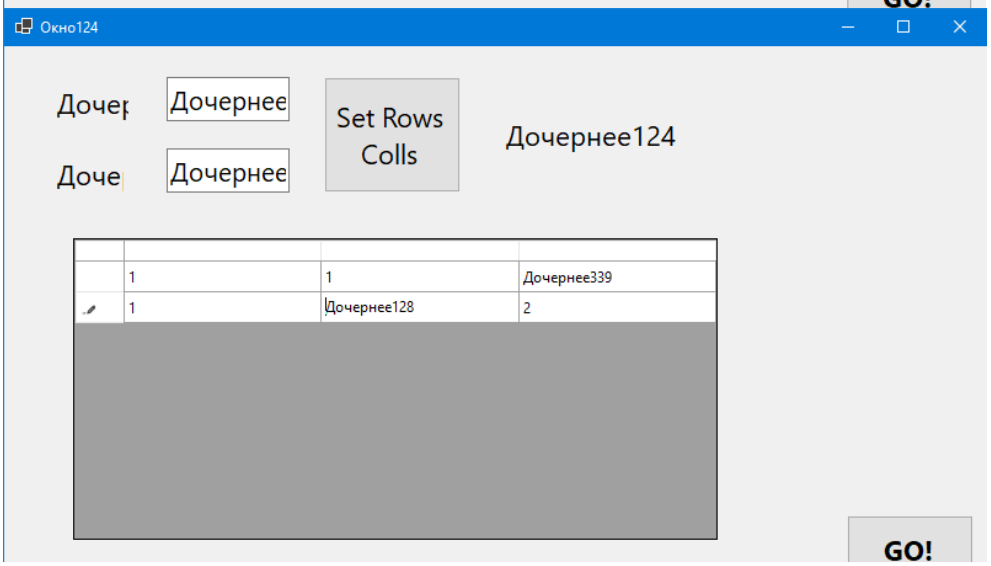




(Здесь в нижнем поле STATIC отображается счетчик событий WM_NCHITTEST, имеющих высокий приоритет, поэтому даже при наведение на шапку окна замененное значение меняется обратно на значение счетчика, на скриншоте ниже – до активации окна)







(В приложении на C# в элементе DataGrid окнами являются только активные ячейки)

Полный код программы доступен здесь: https://pastebin.com/AXUP86eV