Лабораторная работа №3

Управление окнами

А-13а-19 Кутдусов Р.К.

Подготовка к лабораторной работе

1. Функции для работы с окнами.
    - **EnableWindow**(handleW1, FALSE)
      Функция 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.
      Синтаксис

```
BOOL EnableWindow
(
  HWND hWnd,     // дескриптор окна, которое будет включено или заблокировано
  BOOL bEnable   // флаг для включения или отключения ввода информации
);
```

      Если окно было предварительно заблокировано, возвращаемое значение не ноль. Если окно предварительно не было заблокировано, возвращаемое значение нулевое.
    - **IsWindowVisible**
      Функция находит данные о состоянии видимости заданного окна.
      Синтаксис

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

      Если определяемое окно и его родительское окно имеют стиль **WS_VISIBLE**, возвращается значение, отличное от нуля. Если определяемое окно и его родительское окно не имеют стиля **WS_VISIBLE**, возвращается нулевое значение.
    - **ShowWindow**
      Функция устанавливает состояние показа определяемого окна.
      Синтаксис

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

      nCmdShow определяет, как окно должно быть показано
        - **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 актуальном 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.

Если функция завершилась успешно, возвращается значение отличное от нуля. Если функция потерпела неудачу, возвращается нулевое значение.

2. Сообщения для окна
   SendMessage дожидается обработки сообщения и **возвращает результат, PostMessage** просто добавляет сообщение в очередь и совершенно не заботится о том, что произойдет дальше.

```
BOOL WINAPI PostMessage(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);

LRESULT WINAPI SendMessage(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);
```

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 не ищет дочерние окна.

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

Пример: handleWindow=FindWindow(0,"Мое окно");
GetWindow извлекает дескриптор окна, который имеет определенное отношение к заданному окну.

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

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. Она не проверяет окна - потомки.
Пример: handleW2=GetWindow(handleW1,GW_CHILD);
EnumChildWindows

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 **EnumChildWindows** работает до тех пор, пока не будет перечислено последнее дочернее окно или функция повторного вызова не возвратит значение **FALSE**.

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

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

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 **FALSE**.

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

FindWindowEx

Функция **FindWindowEx** извлекает дескриптор окна, имя класса и имя окна которому соответствуют заданные строки.

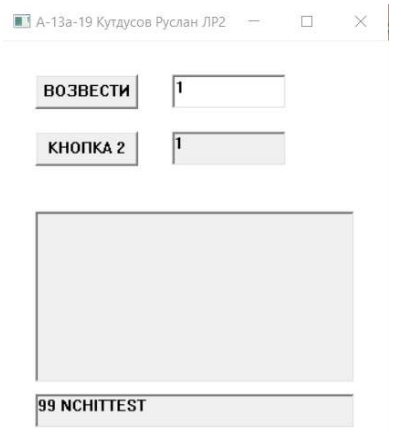
```
HWND FindWindowEx
(
HWND hwndParent,     // дескриптор родительского окна
HWND hwndChildAfter, // дескриптор дочернего окна
LPCTSTR lpszClass,   // указатель имени класса
LPCTSTR lpszWindow   // указатель имени окна
);
```

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

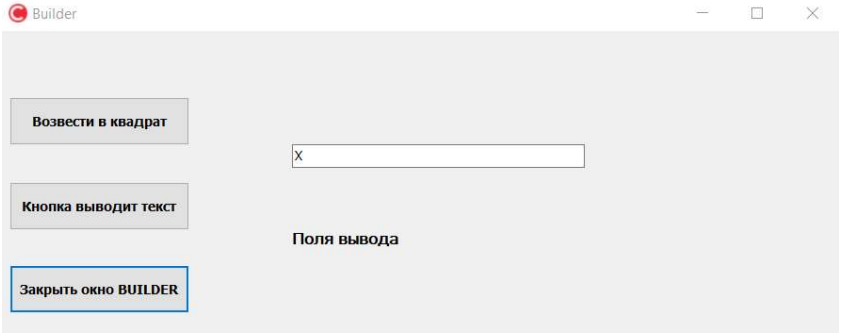
Ход работы

1. Создать основное приложение Win32 Project (оно будет управлять окнами других приложений). Создание проекта → Мастер классических приложений → Далее → Указываем имя проекта, расположение, имя решения → Создать → Тип приложения: Классическое приложение (.exe), Дополнительные параметры: Пустой проект → OK. Добавляем .cpp файл. Добавляем шаблон программы.

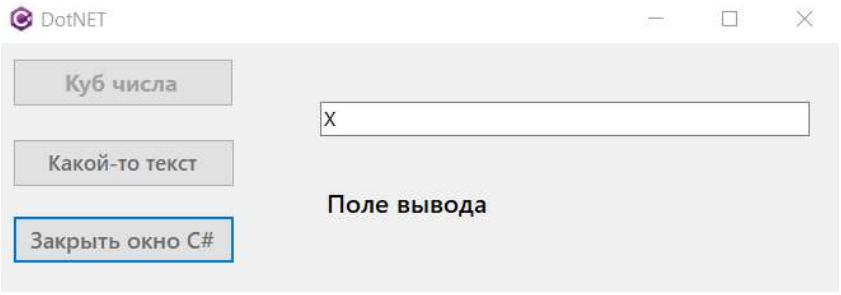
2. Для программных экспериментов необходимы три вспомогательных приложения:
   - Первое: программа из лаб. работы № 2



- 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)

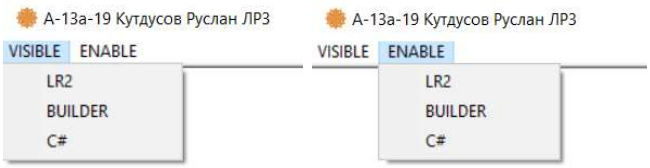


Для приложений добавим в качестве глобальных переменных их дескрипторы:

```
HWND lr2;
HWND builder;
HWND dotnet;
```

3. Добавить в основное приложение следующие возможности:

Чтобы было проще управлять окнами, в основное приложение добавим меню:



- Спрятать окно вспомогательного приложения, если оно видимо

Объявим идентификаторы для пунктов меню:

```
const int IDM_LR2_VIS = 130;
const int IDM_BUILDER_VIS = 135;
const int IDM_DOTNET_VIS = 140;
```

Реализация:

```
case IDM_LR2_VIS:
        if (!lr2)
        { // если дескриптор пустой, то ищем требуемое окно
           lr2 = FindWindow(/*L"DesktopApp2"*/ NULL, L"А-13а-19 Кутдусов Руслан ЛР2");
           if (lr2)
              ShowWindow(lr2, SW_SHOWDEFAULT); // и отображаем, если нашли
        }
        else
            if (!IsWindowVisible(lr2)) ShowWindow(lr2, SW_SHOWDEFAULT);
            else ShowWindow(lr2, SW_HIDE);
        break;
case IDM_BUILDER_VIS:
        if (!builder)
        {
            builder = FindWindow(NULL, L"Builder");
            if (builder)
                 ShowWindow(builder, SW_SHOWDEFAULT);
        }
        else
            if (!IsWindowVisible(builder)) ShowWindow(builder, SW_SHOWDEFAULT);
            else ShowWindow(builder, SW_HIDE);
        break;
case IDM_DOTNET_VIS:
        if (!dotnet)
        {
            dotnet = FindWindow(NULL, L"DotNET");
            if (dotnet)
                 ShowWindow(dotnet, SW_SHOWDEFAULT);
        }
        else
            if (!IsWindowVisible(dotnet)) ShowWindow(dotnet, SW_SHOWDEFAULT);
            else ShowWindow(dotnet, SW_HIDE);
        break;
```

Таким образом, при первом нажатии на соответствующий пункт меню, если окно найдено, оно отображается, при повторном нажатии оно сворачивается.

- Сделать недоступным окно вспомогательного приложения

Реализуем похожим образом. Объявим идентификаторы для пунктов меню:

```
const int IDM_LR2_ENAB = 145;
const int IDM_BUILDER_ENAB = 150;
const int IDM_DOTNET_ENAB = 155;
```

Реализация:

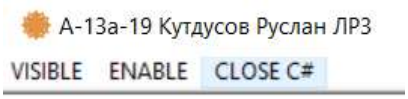
```
case IDM_LR2_ENAB:
        if (!lr2)
        {
            lr2 = FindWindow(/*L"DesktopApp2"*/ NULL, L"А-13а-19 Кутдусов Руслан ЛР2");
            if (lr2)
                 ShowWindow(lr2, SW_SHOWDEFAULT);
        }
        else
            if (!IsWindowEnabled(lr2)) EnableWindow(lr2, true);
            else EnableWindow(lr2, false);
        break;
case IDM_BUILDER_ENAB:
        if (!builder)
        {
            builder = FindWindow(NULL, L"Builder");
            if (builder)
                 ShowWindow(builder, SW_SHOWDEFAULT);
        }
        else
            if (!IsWindowEnabled(builder)) EnableWindow(builder, true);
            else EnableWindow(builder, false);
        break;
case IDM_DOTNET_ENAB:
        if (!dotnet)
        {
            dotnet = FindWindow(NULL, L"DotNET");
            if (dotnet)
                 ShowWindow(dotnet, SW_SHOWDEFAULT);
```

```
        }
        else
            if (!IsWindowEnabled(dotnet)) EnableWindow(dotnet, true);
            else EnableWindow(dotnet, false);
        break;
```

Таким образом, при первом нажатии на соответствующий пункт меню, если окно найдено, оно отображается активным, при повторном нажатии оно становится недоступным.

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

Добавим ещё один пункт меню:




```
const int IDM_DOTNET_CL = 160;
…
case IDM_DOTNET_CL:
        if (dotnet && IsWindowEnabled(dotnet))
        {
            SendMessage(dotnet, WM_DESTROY, 0, 0);
            dotnet = NULL;
        }
        break;
```

4. Управление приложением из лаб. работы № 2:

- Нажать (программно) кнопку и выполнить команду

Возможно три способа.

Добавим в основное приложение кнопку:

```
HWND butt1; const int ID_LR2_POINT1 = 165;
…
butt1 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"), _T("ВОЗВЕСТИ LR2"), WS_VISIBLE |
WS_CHILD, 30, 30, 150, 30, hWnd, (HMENU)ID_LR2_POINT1, hInstance, NULL);
HWND lr2butt;
…
case ID_LR2_POINT1:
        if (lr2 && IsWindowEnabled(lr2)) {
            lr2butt = FindWindowEx(lr2, NULL, NULL, L"ВОЗВЕСТИ");
            if (lr2butt) {
                // 1
                /*SendMessage(lr2butt, WM_LBUTTONDOWN, MK_LBUTTON, 0);
                SendMessage(lr2butt, WM_LBUTTONUP, 0, 0);*/
                // 2
                // SendMessage(lr2butt, BM_CLICK, 0, 0);
                // 3
                SendMessage(lr2butt, BM_SETSTATE, 1, 0);
                Sleep(100);
                SendMessage(lr2, WM_COMMAND, GetWindowLong(lr2butt, GWL_ID), 0);
                SendMessage(lr2butt, BM_SETSTATE, 0, 0);
            }
        }
        break;
```

- Выполнить команду пункта меню

```
HWND butt2; const int ID_LR2_POINT2 = 170;
…
butt2 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"), _T("ABOUT LR2"), WS_VISIBLE | WS_CHILD,
                30, 90, 150, 30, hWnd, (HMENU)ID_LR2_POINT2, hInstance, NULL);
…
HMENU lr2menu;
…
case ID_LR2_POINT2:
        if (lr2 && IsWindowEnabled(lr2))
```

```
            {
            lr2menu = GetMenu(lr2);
            if (lr2menu)
                SendMessage(lr2, WM_COMMAND, GetMenuItemID(lr2menu, 0), 0);
            }
        break;
```

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

```
POINT p;
…
case WM_RBUTTONDOWN:
        if (lr2 && IsWindowEnabled(lr2)) {
            p.x = LOWORD(lParam);
            p.y = HIWORD(lParam);
            ClientToScreen(hWnd, &p);
            ScreenToClient(lr2, &p);
            SendMessage(lr2, WM_RBUTTONDOWN, 0, MAKELONG(p.x, p.y));
        }
        break;
```

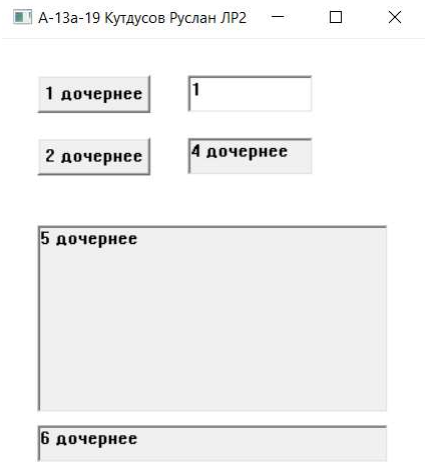
- Заставить окно переместиться по экрану (послав, соответствующие сообщения, так как это окно умеет перемещаться при движении мыши в области клиента)

```
bool window_move = false;
…
POINT p;
…
case WM_LBUTTONDOWN:
        if (lr2 && IsWindowEnabled(lr2))
                {
                window_move = true; // включаем флажок
                p.x = LOWORD(lParam);
                p.y = HIWORD(lParam);
                ClientToScreen(hWnd, &p);
                ScreenToClient(lr2, &p);
                SendMessage(lr2, WM_LBUTTONDOWN, 0, MAKELONG(p.x, p.y));
                }
        break;
case WM_LBUTTONUP:
        window_move = false; // выключаем
        SendMessage(lr2, WM_LBUTTONUP, 0, 0);
        break;
case WM_MOUSEMOVE:
        if (window_move)
                {
                p.x = LOWORD(lParam);
                p.y = HIWORD(lParam);
                ClientToScreen(hWnd, &p);
                ScreenToClient(lr2, &p);
                SendMessage(lr2, WM_MOUSEMOVE, 0, MAKELONG(p.x, p.y));
                }
        break;
```

- Найти и переименовать все дочерние окна вспомогательного приложения

```
int count = 0; // счётчик дочерних окон
…
BOOL CALLBACK RCN(HWND hWnd, LPARAM lParam)
{
        ++count;
        TCHAR buf[100] = { 0 };
        _itoa_s(count, buf, 10);
        strcat_s(buf, _T(" дочернее"));
        SetWindowText(hWnd, buf);
        return TRUE;
}
…
HWND butt3; const int ID_LR2_POINT5 = 175;
…
```

```
butt3 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"), _T("Переименовать LR2"), WS_VISIBLE |
        WS_CHILD, 30, 150, 150, 30, hWnd, (HMENU)ID_LR2_POINT5, hInstance, NULL);
…
case ID_LR2_POINT5:
        if (lr2 && IsWindowEnabled(lr2))
           {
                EnumChildWindows(lr2, &RCN, 0);
                count = 0;
           }
        break;
```



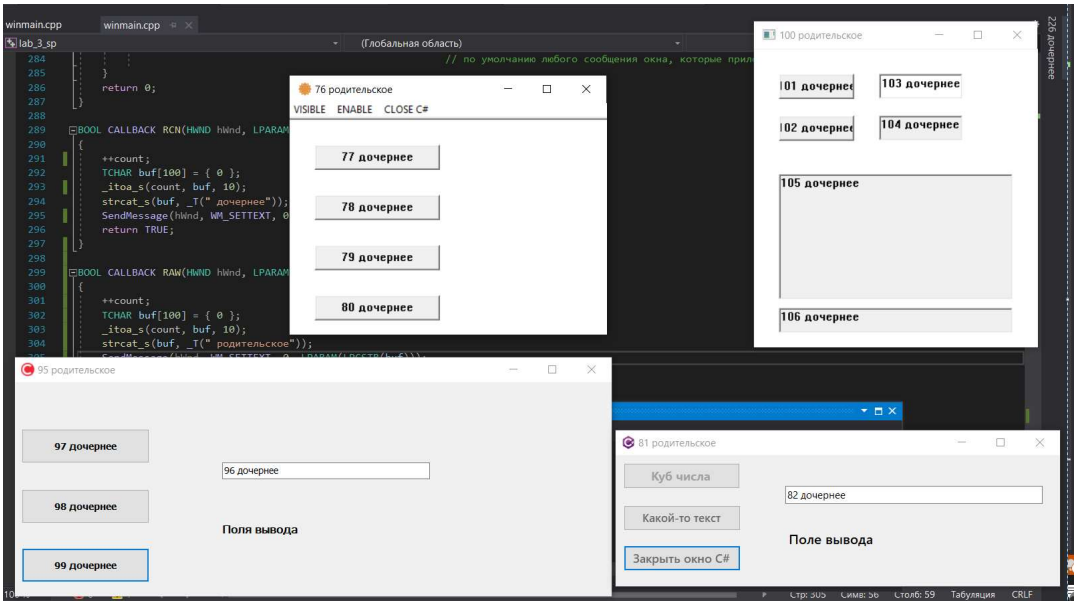
3 дочернее – EDIT

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.).

```
HWND butt4; const int ID_RENAMEALL = 180;
…
butt4 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"), _T("Переименовать ВСЁ"), WS_VISIBLE |
WS_CHILD, 30, 210, 150, 30, hWnd, (HMENU)ID_RENAMEALL, hInstance, NULL);
…
BOOL CALLBACK RAW(HWND hWnd, LPARAM lParam)
{
        ++count;
        TCHAR buf[100] = { 0 };
        _itoa_s(count, buf, 10);
        strcat_s(buf, _T(" родительское"));
        SetWindowText(hWnd, buf);
        EnumChildWindows(hWnd, &RCN, 0);
        return TRUE;
}
…
case ID_RENAMEALL:
        EnumWindows(&RAW, 0);
        count = 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).

   Визуально окон чуть больше пятисот. Замечу, что поля label в Builder и в Windows Forms не являются окнами, а поля edit в обоих случаях являются окнами. Также не являются окнами все кнопки Windows Forms.


winmain.cpp
winmain.cpp
lab_3_sp
(Глобальная область)
// по умолчанию любого сообщения окна, которые прил
}
return 0;
}
BOOL CALLBACK RCN(HWND hWnd, LPARAM
{
++count;
TCHAR buf[100] = { 0 };
_itoa_s(count, buf, 10);
strcat_s(buf, _T(" дочернее"));
SendMessage(hWnd, WM_SETTEXT, 0
return TRUE;
}
BOOL CALLBACK RAW(HWND hWnd, LPARAM
{
++count;
TCHAR buf[100] = { 0 };
_itoa_s(count, buf, 10);
strcat_s(buf, _T(" родительское"));
76 родительское
VISIBLE
ENABLE
CLOSE C#
77 дочернее
78 дочернее
79 дочернее
80 дочернее
100 родительское
101 дочернее
103 дочернее
102 дочернее
104 дочернее
105 дочернее
106 дочернее
226 дочернее
95 родительское
97 дочернее
96 дочернее
98 дочернее
Поля вывода
99 дочернее
81 родительское
Куб числа
82 дочернее
Какой-то текст
Поле вывода
Закрыть окно C#
Стр: 305
Симв: 56
Столб: 59
Табуляция
CRLF