Лабораторная работа №1

Структура приложения под Windows

А-13а-19 Кутдусов Р.К.

Подготовка к лабораторной работе

1. Получить координаты окна (прямоугольник) - функция **`GetWindowRect`**

   Функционал: функция извлекает размеры рамки, ограничивающей прямоугольник заданного окна. Размеры даются в экранных координатах, которые отсчитываются относительно верхнего левого угла экрана.

   Синтаксис:

```
BOOL GetWindowRect(HWND hWnd, LPRECT lpRect);
```

   Параметры:

   `hWnd` [in] дескриптор окна
   `lpRect` [out] указатель на структуру, которая принимает экранные координаты левого верхнего и нижнего правого углов окна

   Возвращаемое значение: *`TRUE(1)/FALSE(0)`*

2. Установить координаты окна - функция **`SetWindowPos`**

   Функционал: функция изменяет размер, позицию и **z-последовательность** (обозначает позицию окна в стеке перекрывающихся окон) дочернего, выскакивающего или окна верхнего уровня. Дочерние, выскакивающие и окна верхнего уровня размещаются по порядку согласно их появлению на экране. Самое верхнее окно принимает самый высокий ранг и становится первым окном в **z-последовательности**.

   Синтаксис:

```
BOOL SetWindowPos(HWND hWnd, HWND hWndInsertAfter, int x, int y, int cx, int cy, UINT uFlags);
```

   Параметры:

   `hWnd` [in] дескриптор окна
   `hWndInsertAfter` [in] дескриптор окна, предшествующего установленному окну в z-последовательности
   `x` [in] устанавливает новую позицию с левой стороны окна (в рабочих координатах)
   `y` [in] устанавливает новую позицию верхней части окна (в рабочих координатах)
   `cx` [in] устанавливает новую ширину окна (в пикселях)
   `cy` [in] устанавливает новую высоту окна (в пикселях)
   `uFlags` [in] оперделяет флажки, устанавливающие размеры и позицию окна

   Возвращаемое значение: *`TRUE(1)/FALSE(0)`*

3. Переместить окно – функция **`MoveWindow`**

Функционал: функция изменяет позицию и габариты определяемого окна. Для окна верхнего уровня позиция и габариты отсчитываются относительно левого верхнего угла экрана. Для дочернего окна габариты и позиция отсчитываются относительно левого верхнего угла рабочей области родительского окна.

Синтаксис:

```
BOOL MoveWindow(HWND hWnd, int x, int y, int nWidth, int nHeight, BOOL bRepaint);
```

Параметры:
hWnd [in] дескриптор окна
x [in] устанавливает новую позицию левой стороны окна
y [in] устанавливает новую позицию верхней части окна
nWidth [in] устанавливает новую ширину окна
nHeight [in] устанавливает новую высоту окна
bRepaint [in] определяет, должно ли окно быть перерисовано

Возвращаемое значение: *TRUE(1)/FALSE(0)*

4. Вывод сообщения – функция **MessageBox**

Функционал: функция создает окно сообщения, показывает его на экране и использует в дальнейшем. Это окно содержит определяемое программой сообщение и заголовок.

Синтаксис:

```
int MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType);
```

Параметры:

hWnd [in] дескриптор окна владельца, которое создает окно сообщения. Если этот параметр - NULL, окно сообщения не имеет окна владельца

lpText [in] указатель на символьную строку с нулем в конце, которая содержит сообщение показываемое на экране
lpCaption [in] указатель на символьную строку с нулем в конце, которая содержит заголовок диалогового окна (окна сообщения). Если этот параметр - NULL, используется заданный по умолчанию заголовок **Error** (Ошибка).
uType [in] устанавливает содержание и режим работы диалогового окна

Возвращаемое значение: если окно сообщения имеет кнопку **Cancel**, то функция возвращает значение **IDCANCEL**, или если обрабатывается клавиша **ESC**, или выбрана кнопка **Cancel**. Если окно сообщения не имеет кнопки **Cancel**, **ESC** не имеет никакого действия. Если функция завершается ошибкой, возвращаемое значение равняется нулю.

5. Создать окно - функция **CreateWindow**

Функционал: функция создает перекрывающее, выскакивающее или дочернее окно. Она определяет класс, заголовок, стиль окна и начальную позицию, размер окна. Функция также определяет и окно родителя или владельца, если таковые имеются и меню окна.

Синтаксис:

```
HWND CreateWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName, DWORD dwStyle, int x,
int y, int nWidth, int nHeight, HWND hWndParent, HMENU hMenu, HANDLE hInstance,
LPVOID lpParam);
```

Параметры:

lpClassName [in] указывает на строку с нулевым символом в конце или на атом класса, созданный предыдущим вызовом функции **RegisterClass** или **RegisterClassEx**
lpWindowName [in] указывает на строку с нулевым символом на конце, которая определяет имя окна
dwStyle [in] определяет стиль создаваемого окна
x [in] определяет начальную горизонтальную позицию окна
y [in] определяет начальную вертикальную позицию окна
nWidth [in] определяет ширину окна (в единицах измерения для устройства)
nHeight [in] определяет высоту окна (в единицах измерения устройства)
hWndParent [in] дескриптор окна родителя или владельца создаваемого окна
hMenu [in] дескриптор меню или определяет идентификатор дочернего окна в зависимости от стиля окна
hInstance [in] дескриптор экземпляра модуля, который будет связан с окном
lpParam [in] указывает на значение, переданное окну через структуру **CREATESTRUCT**, переданную в параметре *lParam* сообщения **WM_CREATE** (описана позже). Если прикладная программа вызывает **CreateWindow**, чтобы создать рабочее окно многодокументного интерфейса (**MDI**), *lpParam* должен указывать на структуру **CLIENTCREATESTRUCT**

Возвращаемое значение: если функция завершается успешно, возвращаемое значение - дескриптор созданного окна. Если функция завершилась ошибкой, возвращаемое значение - NULL.

6. Установка идентификатора – функция **SetWindowLong**

Функционал: функция заменяет атрибуты указанного окна. Функция также устанавливает и 32-разрядное (long) значение при заданном смещении в дополнительную память окна.

Синтаксис:

```
LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);
```

Параметры:

hWnd [in] дескриптор окна и, косвенно, класс к которому принадлежит окно
nIndex [in] определяет отсчитываемое от нуля смещение устанавливаемого значения
dwNewLong [in] устанавливает заменяемое значение

Возвращаемое значение: если функция завершается успешно, возвращаемое значение - предыдущая величина указанного 32-разрядного целого числа.
Если функция завершается ошибкой, возвращаемое значение равняется нулю.

7. Функции **GetWindowText** и **SetWindowText**

Функционал: Функция **GetWindowText** копирует текст заголовка определяемого окна (если окно имеет его) в буфер. Если заданное окно является органом управления, копируется его текст. Однако функция **GetWindowText** не может извлекать текст органа управления в другом приложении.

Функция **SetWindowText** изменяет текст заголовка заданного окна (если таковой имеется). Если определяемое окно - орган управления, то изменяется его текст. Однако **SetWindowText** также не может изменить текст органа управления в другом приложении.

Синтаксис:

```
int GetWindowText(HWND hWnd, LPTSTR lpString, int nMaxCount);

BOOL SetWindowText(HWND hWnd, LPCTSTR lpString);
```

Параметры:

GetWindowText

hWnd [in] дескриптор окна или органа управления, содержащего текст
lpString [out] указывает на буфер, который примет текст. Если строка является такой же длины или длиннее, чем буфер, она обрезается и завершается символом NULL
nMaxCount устанавливает максимальное число символов для копирования в буфер, включая символ NULL. Если текст превышает это ограничение, он усекается

SetWindowText

hWnd [in] дескриптор окна или органа управления, текст которого должен быть изменен
lpString[in] указатель на строку с нулевым символом в конце, которую нужно использовать как новый заголовок или текст органа управления

Возвращаемое значение:

GetWindowText

Если функция завершается успешно, возвращаемое значение - длина, в символах, скопированной строки, не, включая символа конца строки (нуль-терминатора). Если у окна нет заголовка или текста, если строка заголовка - пустая строка или, если дескриптор окна или органа управления недопустимы, возвращаемое значение нулевое.

SetWindowText

TRUE(1)/FALSE(0)

8. Функции **SetParent** и **GetParent**

Функционал: Функция **SetParent** заменяет родительское окно заданного дочернего окна. Функция **GetParent** извлекает дескриптор родителя или владельца заданного окна.

Синтаксис:

```
HWND SetParent(HWND hWndChild, HWND hWndNewParent);
HWND GetParent(HWND hWnd);
```

Параметры:

SetParent

hWndChild [in] дескриптор дочернего окна
hWndNewParent [in] дескриптор нового родительского окна. Если этот параметр - NULL, окно рабочего стола становится новым родительским окном

GetParent

hWnd [in] дескриптор окна, дескриптор родительского окна которого должен быть найден

Возвращаемое значение:

SetParent

Если функция завершается успешно, возвращаемое значение - дескриптор предыдущего родительского окна. Если функция не выполняет задачу, возвращаемое значение - NULL.

GetParent

Если окно - дочернее окно, величина возвращаемого значения - дескриптор родительского окна. Если окно - окно верхнего уровня, величина возвращаемого значения - дескриптор окну владельца. Если окно - не имеющее владельца окно верхнего уровня или если функция завершается с ошибкой, величина возвращаемого значения - NULL.

9. Функция **Sleep** (пауза)

Функционал: функция приостанавливает работу по выполнению текущего потока на заданный промежуток времени.

Синтаксис:

```
VOID Sleep(DWORD dwMilliseconds);
```

Параметры:

dwMilliseconds [in] минимальный интервал времени, в миллисекундах, на которое приостанавливается выполняемая работа. Значение INFINITE вызывает бесконечную задержку

Возвращаемое значение: *VOID*

10. Функции для работы с регионами (графический объект "регион" определяет плоскую произвольную область)

Функционал:

Функция **CombineRgn** - комбинирует два региона между собой
Функция **CreateEllipticRgn** - создает регион в виде эллипса или окружности
Функция **CreatePolygonRgn** - создает регион в виде многоугольника
Функция **CreateRectRgn** - создает прямоугольный регион
Функция **CreateRoundRectRgn** - создает регион со скругленными краями из прямоугольной области
Функция **SetWindowRgn** - прикрепляет регион к указанному окну

Синтаксис:

```
int CombineRgn(HRGN hrgnDst, HRGN hrgnSrc1, HRGN hrgnSrc2, int  iMode);
HRGN CreateEllipticRgn(int x1, int y1, int x2, int y2);
HRGN CreatePolygonRgn(const POINT *ppt1, int         cPoint, int         iMode);
HRGN CreateRectRgn(int x1, int y1, int x2, int y2);
HRGN CreateRoundRectRgn(int x1, int y1, int x2, int y2, int w, int h);
int SetWindowRgn(HWND hWnd, HRGN hRgn, BOOL bRedraw);
```

Параметры:

CombineRgn

hrgnDst [in] дескриптор получаемого региона. Регион уже должен существовать перед вызовом функции

hrgnSrc1 [in] первый исходный регион

hrgnSrc2 [in] второй исходный регион
iMode [in] флаг, определяющий способ комбинирования регионов

CreateEllipticRgn

x1 [in] координата x верхнего левого угла ограничительного прямоугольника (самый маленький возможный прямоугольник, который может соответствовать эллипсу)
y1 [in] координата y верхнего левого угла ограничительного прямоугольника
x2 [in] координата x нижнего правого угла ограничительного прямоугольника
y2 [in] координата y нижнего правого угла ограничительного прямоугольника

CreatePolygonRgn

ppt1 [in] массив точек, определяющих вершины многоугольника
cPoint [in] число элементов в массиве
iMode [in] один из следующих флагов, определяющих режим заполнения многоугольника
ALTERNATE = 1 выбор между заполнением и незаполнением непрерывных секций, чьи границы определены краями многоугольника, пересекающегося через внутреннюю область многоугольника
WINDING = 2 любая секция внутри многоугольника заполнена, независимо от любых внутренних многоугольных границ и граней

CreateRectRgn

x1 [in] координата x левого верхнего угла прямоугольника
y1 [in] координата y левого верхнего угла прямоугольника
x2 [in] координата x нижнего правого угла прямоугольника
y2 [in] координата y нижнего правого угла прямоугольника

CreateRoundRectRgn

x1 [in] определяет x -координату верхнего левого угла области
y1 [in] определяет y - координату верхнего левого угла области
x2 [in] определяет x -координату нижнего правого угла области
y2 [in] определяет y -координату нижнего правого угла области
w [in] определяет ширину эллипса, используемого для создания закругленных углов
h [in] определяет высоту эллипса, используемого для создания закругленных углов

SetWindowRgn

hWnd [in] дескриптор окна, регион окна которого должен быть установлен
hRgn [in] дескриптор региона. Функция устанавливает регион окна в окне для этого региона. Если значение NULL, функция устанавливает регион окна в NULL
bRedraw [in] определяет, перерисовывает ли система окно после установки региона окна Если параметр - TRUE, система это выполняет; в противном случае она этого не делает. Как правило, параметр устанавливается как TRUE, если окно видимо

Возвращаемое значение:

CombineRgn

Функция возвращает одну из следующих констант, определяющих результат комбинирования:
ERROR = *0*
Ошибка при попытке комбинирования регионов
NULLREGION = *1*
Полученный регион пуст
SIMPLEREGION = *2*
Полученный регион в форме прямоугольника
COMPLEXREGION = *3*
Полученный регион, содержащий более одного прямоугольника

CreateEllipticRgn

Функция возвращает дескриптор созданной области в успешном случае или 0 в случае ошибки.

CreatePolygonRgn

Функция возвращает дескриптор созданной области в успешном случае или 0 в случае ошибки.

CreateRectRgn

Функция возвращает дескриптор созданной области в успешном случае или 0 в случае ошибки.

CreateRoundRectRgn

Функция возвращает дескриптор созданной области в успешном случае или 0 в случае ошибки.

SetWindowRgn

Если функция завершается успешно, возвращаемое значение - не нуль. Если функция завершается ошибкой, возвращаемое значение - нуль.

11. События мыши, события **WM_CREATE**, **WM_COMMAND**, **WM_DESTROY** и **WM_QUIT**

WM_RBUTTONDBLCLK (WM_LBUTTONDBLCLK, WM_MBUTTONDBLCLK)

Сообщение помещается в очередь, если пользователь дважды щелкает правой (левой, средней) кнопкой мыши, в то время, когда курсор находится в рабочей области окна. Если мышь не захвачена, сообщение помещается в окно под курсором. В противном случае сообщение помещается в окно, которое захватило мышь.

```
WM_RBUTTONDBLCLK
    WPARAM wParam
    LPARAM lParam;
WM_LBUTTONDBLCLK
    WPARAM wParam
    LPARAM lParam;
WM_MBUTTONDBLCLK
    WPARAM wParam
    LPARAM lParam;
```

wParam указывает, находятся ли в нажатом состоянии различные виртуальные клавиши

lParam младшее слово устанавливает x-координату курсора. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области
Если приложение обрабатывает это сообщение, оно должно возвратить 0.

WM_RBUTTONDOWN (WM_LBUTTONDOWN, WM_MBUTTONDOWN)

Сообщение посылается тогда, когда пользователь нажимает правую (левую, среднюю) кнопку мыши, в то время, когда курсор находится в рабочей области окна. Если мышь не захвачена, сообщение посылается в окно под курсором. В противном случае сообщение помещается в окно, которое захватило мышь.

```
WM_RBUTTONDOWN
    WPARAM wParam
    LPARAM lParam;
WM_LBUTTONDOWN
    WPARAM wParam
    LPARAM lParam;
WM_MBUTTONDOWN
    WPARAM wParam
    LPARAM lParam;
```

wParam указывает, находятся ли в нажатом состоянии различные виртуальные клавиши

lParam младшее слово устанавливает x-координату курсора. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области
Если приложение обрабатывает это сообщение, оно должно возвратить 0.

WM_RBUTTONUP (WM_LBUTTONUP, WM_MBUTTONUP)

Сообщение посылается тогда, когда пользователь отпускает правую (левую, среднюю) кнопку мыши, в то время, когда курсор находится в рабочей области окна. Если мышь не захвачена, сообщение посылается в окно под курсором. В противном случае сообщение помещается в окно, которое захватило мышь.

```
WM_RBUTTONUP
    WPARAM wParam
    LPARAM lParam;
WM_LBUTTONUP
    WPARAM wParam
    LPARAM lParam;
WM_MBUTTONUP
    WPARAM wParam
    LPARAM lParam;
```

wParam указывает, находятся ли в нажатом состоянии различные виртуальные клавиши

lParam младшее слово устанавливает x-координату курсора. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области
Если приложение обрабатывает это сообщение, оно должно возвратить 0.

WM_CAPTURECHANGED

Сообщение отправляется в окно, которое теряет захват мыши.

```
WM_CAPTURECHANGED
    WPARAM wParam
    LPARAM lParam;
```

wParam этот параметр не используется
lParam дескриптор окна, получающего захват мыши
Если приложение обрабатывает это сообщение, оно должно возвратить 0.

WM_MOUSEACTIVATE

Сообщение отправляется тогда, когда курсор находится в неактивном окне, а пользователь нажимает кнопку мыши.

```
WM_MOUSEACTIVATE
    WPARAM wParam
    LPARAM lParam;
```

wParam дескриптор родительского окна верхнего уровня окна, которое становится активным
lParam младшее слово устанавливает значение местоположения курсора, возвращенное функцией **DefWindowProc** в результате обработки сообщения **WM_NCHITTEST**. Старшее слово устанавливает идентификатор созданного сообщения, когда пользователь нажал кнопку мыши. Сообщение мыши или сбрасывается, или посылается в окно, в зависимости от возвращаемого значения
Возвращаемое значение устанавливает, должно ли окно становиться активным, и должен ли идентификатор сообщения мыши сбрасываться.

WM_NCHITTEST

Сообщение отправляется в окно тогда, когда перемещается курсор, или когда кнопка мыши нажимается или отпускается. Если мышь не захвачена, сообщение отправляется в окно под курсором. В противном случае сообщение отправляется в окно, которое захватило мышь.

WM_NCHITTEST

WPARAM *wParam*
LPARAM *lParam;*

wParam этот параметр не используется
lParam младшее слово устанавливает x-координату курсора. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области
Возвращаемое значение функцией **DefWindowProc** — это одно из значений, которое указывает позицию острия курсора

WM_MOUSEHOVER

Сообщение посылается в окно, когда курсор нависает над рабочей областью окна в течение определенного периода времени.

WM_MOUSEHOVER
WPARAM *wParam*
LPARAM *lParam;*

wParam указывает, находятся ли в нажатом состоянии различные виртуальные клавиши
lParam младшее слово устанавливает x-координату курсора. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области
Если приложение обрабатывает это сообщение, оно должно возвратить 0.

WM_MOUSELEAVE

Сообщение посылается в окно тогда, когда курсор оставляет рабочую область окна.

WM_MOUSELEAVE
WPARAM *wParam*
LPARAM *lParam;*

wParam не используется; должен быть нуль
lParam не используется; должен быть нуль
Если приложение обрабатывает это сообщение, оно должно возвратить 0.

WM_MOUSEWHEEL

Сообщение отправляется в окно с фокусом, когда прокручивается колесико мыши.

WM_MOUSEWHEEL
WPARAM *wParam*
LPARAM *lParam;*

wParam старшее слово указывает интервал, на который прокрутилось колесико, выраженный в нескольких или раздельных **WHEEL_DELTA**, число которых - 120. Положительное значение указывает, что колесико вращалось вперед, в сторону от пользователя; отрицательное значение указывает, что колесико вращалось назад, к пользователю. Младшее слово указывает, находятся ли в нажатом состоянии различные виртуальные клавиши.
lParam младшее слово устанавливает x-координату указателя, относительно левого верхнего угла экрана. Старшее слово устанавливает y-координату указателя, относительно левого верхнего угла экрана.
Если приложение обрабатывает это сообщение, оно должно возвратить 0.

WM_CREATE

Сообщение отправляется тогда, когда программа запрашивает, какое окно будет создаваться вызовом функции **CreateWindowEx** или **CreateWindow.** **(**Сообщение посылается перед возвращением значения функцией). Оконная процедура нового окна принимает это сообщение после того, как окно создано, но до того, как окно становится видимым.

WM_CREATE

WPARAM *wParam*
LPARAM *lParam*;

wParam этот параметр не используется

lParam указатель на структуру **CREATESTRUCT**, которая содержит информацию о создаваемом окне

Если приложение обрабатывает это сообщение, оно должно возвратить 0, чтобы продолжить создание окна. Если прикладная программа возвращает (-1), то окно разрушается, и функция **CreateWindowEx** или **CreateWindow** возвращает значение дескриптора NULL.

WM_COMMAND

Сообщение посылается когда:

- производится выбор пункта меню
- элемент управления посылает уведомительное сообщение родительскому окну
- происходит нажатие клавиши акселератора.

Обработка этого сообщения производится в главной функции окна.

WM_COMMAND
WPARAM *wParam*
LPARAM *lParam*;

wParam определяет источник сообщения: элемент управления или акселератор.

lParam идентификатор элемента, если это не акселератор.

После обработки этого сообщения необходимо вернуть 0.

WM_DESTROY

Сообщение отправляется тогда, когда окно разрушается. Оно отправляется оконной процедуре разрушаемого окна после того, как окно удаляется с экрана.

Это сообщение отправляется сначала разрушаемому окну, а затем дочерним окнам (если таковые имеются), когда они разрушаются. В ходе обработки сообщения оно может быть принято, так как все дочерние окна все еще существуют.

WM_DESTROY
WPARAM *wParam*
LPARAM *lParam*;

wParam этот параметр не используется

lParam этот параметр не используется

Если программа обрабатывает это сообщение, она должно возвратить 0.в

WM_QUIT

Сообщение указывает запрос на завершение работы приложения и создается, когда из прикладной программы вызывается функция **PostQuitMessage**. Это вынуждает функцию **GetMessage** возвратить 0.

WM_QUIT
WPARAM *wParam*
LPARAM *lParam*;

wParam определяет код выхода из программы, данный в функции **PostQuitMessage**.

lParam этот параметр не используется.

Это сообщение не имеет возвращаемого значения, потому что оно принуждает цикл сообщений завершить работу до того, как сообщение отправляется оконной процедуре прикладной программы.

Ход работы

1. Создаём приложение Win32 Project (в Visual Studio 2019)
   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.
2. Работаем с ресурсами приложения:
   - Изменим заголовок окна
     Объявим переменную для имени заголовка

```
// The string that appears in the application's title bar.
static TCHAR szTitle[] = L"А-13а-19 Кутдусов Руслан ЛР1";
```

   - Изменим иконку приложения
     В структуре **WNDCLASSEX**, которая содержит информацию о классе окна, есть поля, которые задают иконки для приложения. Иконки загрузим с помощью функции **LoadImage**

```
WNDCLASSEX wcex;
wcex.hInstance      = hInstance; // первый параметр функции WinMain - дескриптор  текущего
                                 // экземпляра окна
wcex.hIcon          = (HICON)LoadImage(wcex.hInstance, L"icon2.ico", IMAGE_ICON, 0, 0,
LR_LOADFROMFILE); // больший значок
wcex.hIconSm        = (HICON)LoadImage(wcex.hInstance, L"icon1.ico", IMAGE_ICON, 0, 0,
LR_LOADFROMFILE); // малый значок
```

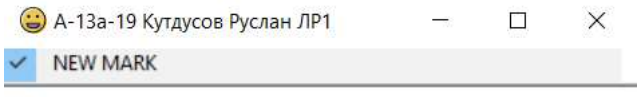
   - Добавим в меню новый пункт
     Создаем пустое меню с помощью функции **CreatePopupMenu**, затем добавляем туда новый пункт с помощью функции **AppendMenu**

```
HMENU hmn = CreatePopupMenu();
const int UNIQUE_ID = 101;
AppendMenu(hmn, MF_CHECKED, UNIQUE_ID, L"NEW MARK");
```

     В итоге все это объединяем в функции **CreateWindowEx**

```
HWND hWnd = CreateWindowEx(
      WS_EX_OVERLAPPEDWINDOW,       // an optional extended window style
      szWindowClass,                // the name of the application
      szTitle,                      // the text that appears in the title bar
      WS_OVERLAPPEDWINDOW,          // the type of window to create
      50, 50,                       // initial position (x, y)
      400, 300,                     // initial size (width, length)
      NULL,                         // the parent of this window
      hmn,                          // menu bar
      hInstance,                    // the first parameter from WinMain
      NULL                          // not used in this application
          );
```

     Получаем окно следующего вида:



3. Также в функции **CreateWindowEx** мы задали координаты левого верхнего угла и размеры запускаемого окна.
   Для изменения координат и размера окна после его создания можно использовать функцию **MoveWindow**. Это можно сделать при создании окна (**WM_CREATE**).

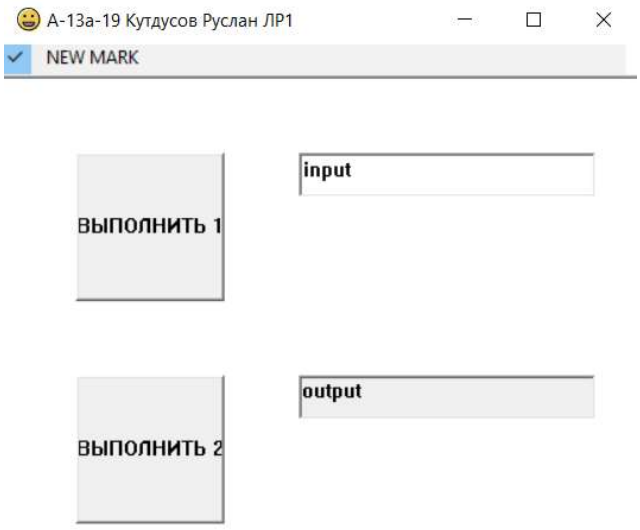
```
MoveWindow(
       hWnd, // дескриптор окна
       100,  // новая позиция по горизонтали
       100,  // новая позиция по вертикали
       450,  // новая ширина
       400,  // новая высота
       FALSE // флажок перекраски
       );
```

4. Создадим управляющие элементы пользовательского интерфейса с помощью **CreateWindowEx**

```
// уникальные идентификатор управляющих элементов
   const int UQID_BUTT1 = 110;
   const int UQID_BUTT2 = 111;
   const int UQID_ED1   = 120;
   const int UQID_ST1   = 125;

   HWND butt1 = CreateWindowEx(WS_EX_WINDOWEDGE, L"BUTTON", L"ВЫПОЛНИТЬ 1", WS_VISIBLE |
WS_CHILD, 50, 50, 100, 100, hWnd, (HMENU)UQID_BUTT1, hInstance, NULL); // кнопка 1
   HWND butt2 = CreateWindowEx(WS_EX_WINDOWEDGE, L"BUTTON", L"ВЫПОЛНИТЬ 2", WS_VISIBLE |
WS_CHILD, 50, 200, 100, 100, hWnd, (HMENU)UQID_BUTT2, hInstance, NULL); // кнопка 2
   HWND ed1   = CreateWindowEx(WS_EX_CLIENTEDGE, L"EDIT", L"input", WS_VISIBLE | WS_CHILD |
ES_LEFT, 200, 50, 200, 30, hWnd, (HMENU)UQID_ED1, hInstance, NULL); // поля ввода
   HWND st1   = CreateWindowEx(WS_EX_CLIENTEDGE, L"STATIC", L"output", WS_VISIBLE | WS_CHILD |
ES_LEFT, 200, 200, 200, 30, hWnd, (HMENU)UQID_ST1, hInstance, NULL); // поле вывода
```

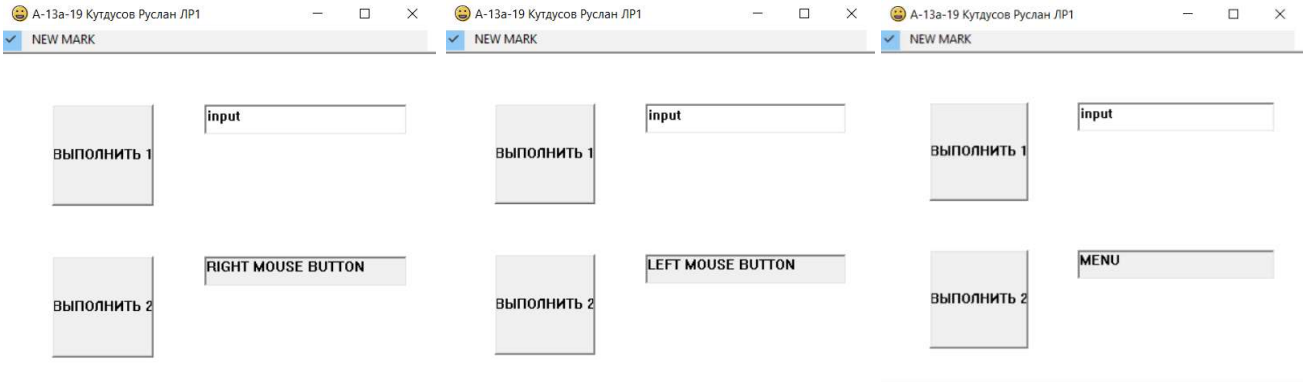
Получаем окно следующего вида:



5. Напишем обработчики событий (внутри **WndProc**) для кнопок, пункта меню и некоторых событий
   - События: WM_CREATE, WM_DESTROY, нажатие левой кнопки мыши, нажатие правой кнопки мыши, щелчок на добавленном пункте меню должны выводить сообщения о том, какое именно событие произошло

```
case WM_CREATE:
      SetWindowText(st1, _T("CREATE"));
      MoveWindow(
          hWnd, // дескриптор окна
          100,  // новая позиция по горизонтали
          100,  // новая позиция по вертикали
          450,  // новая ширина
```

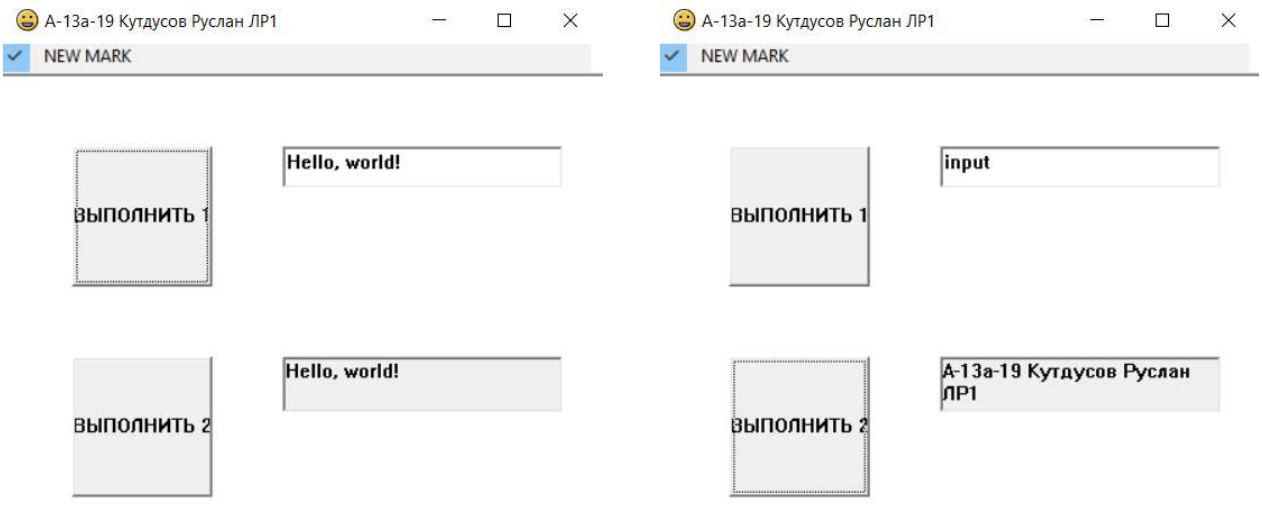
```
            400,  // новая высота
            FALSE // флажок перекраски
        );
        break;
case WM_RBUTTONDOWN:
        SetWindowText(st1, _T("RIGHT MOUSE BUTTON"));
        SetParent(butt1, hWnd);
        MoveWindow(butt1, 50, 50, 100, 100, FALSE);
        break;
case WM_LBUTTONDOWN:
        SetWindowText(st1, _T("LEFT MOUSE BUTTON"));
        SetWindowPos(butt1, HWND_TOP, 0, 0, 100, 100, SWP_SHOWWINDOW);
        SetParent(butt1, NULL);
        ButtonMove(butt1);
case WM_DESTROY:
        SetWindowText(st1, _T("DESTROY"));
        PostQuitMessage(0);
        break;
```



- Первая кнопка BUTTON: текст, введенный в поле EDIT должен появиться в поле STATIC
- Вторая кнопка BUTTON: в поле STATIC должен быть выведен заголовок окна

```
TCHAR buf[100] = { 0 };
…

case WM_COMMAND:
        switch (wParam)
        {
        case UQID_BUTT1: // первая кнопка
            GetWindowText(ed1, buf, 100); // читаем из поля ввода
            SetWindowText(st1, buf);      // записываем в поле вывода
            break;
        case UQID_BUTT2: // вторая кнопка
            GetWindowText(hWnd, buf, 100); // читаем заголовок главного окна
            SetWindowText(st1, buf); // записываем заголовок в поле вывода
            break;
        case UNIQUE_ID:
            SetWindowText(st1, _T("NEW MARK")); // кликаем на пункт меню
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        break;
```



6. Обработчик пункта меню должен: сменить окно - родителя у первой кнопки, а затем заставить ее медленно перемещаться по экрану

```
case WM_RBUTTONDOWN:
      SetWindowText(st1, _T("RIGHT MOUSE BUTTON"));
      SetParent(butt1, hWnd);
      MoveWindow(butt1, 50, 50, 100, 100, FALSE);
      break;
case WM_LBUTTONDOWN:
      SetWindowText(st1, _T("LEFT MOUSE BUTTON"));
      SetWindowPos(butt1, HWND_TOP, 0, 0, 100, 100, SWP_SHOWWINDOW);
      SetParent(butt1, NULL);
      ButtonMove(butt1);
       break;
```

Нажатие правой кнопки мыши должно обеспечить возвращение кнопки в окно программы. Функция **ButtonMove**

```
void ButtonMove(HWND button)
{
    int i = 0;
    for (; i <= 750; ++i)
    {
      SetWindowText(button, L"Выполнить 1");
      MoveWindow(button, i, i, 100, 100, FALSE);
      Sleep(10);

    }
    for (; i >= 0; --i)
    {
      SetWindowText(button, L"Выполнить 1");
      MoveWindow(button, i, i, 100, 100, FALSE);
      Sleep(10);
    }
}
```

7. Оформить окно приложения в форме региона с помощью функций работы с регионами: прямоугольник с эллипсом (нижний край окна) и вырезанным посередине кругом (расположить все управляющие элементы, чтобы они были видны).

```
RECT WRect;
   const int g_1 = 150;
   const int g_2 = 100;
   GetWindowRect(hWnd, &WRect); // получим координаты окна
   HRGN Rgn1 = CreateEllipticRgn(-g_1, -g_1, WRect.right - WRect.left + g_1, WRect.bottom -
WRect.top);
   HRGN Rgn2 = CreateEllipticRgn(g_1 + g_2, g_1, WRect.bottom - WRect.top - g_1 + g_2, WRect.bottom
- WRect.top - g_1); // кружок
   CombineRgn(Rgn1, Rgn1, Rgn2, RGN_XOR); // наложение регионов друг на друга и удаление кружка
SetWindowRgn(hWnd, Rgn1, FALSE);
```