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

Отчет к лабораторной работе № 1

Дисциплина: «Системное программирование»

Михаил Чуворкин
12.9.2021

Подготовка к лабораторной работе

GetWindowRect - Извлекает размеры ограничивающего прямоугольника указанного окна. Размеры даны в координатах экрана относительно левого верхнего угла экрана.

```
BOOL GetWindowRect(
  HWND   hWnd, // Дескриптор окна
  LPRECT lpRect // Указатель на структуру RECT, которая получает экранные координаты левого верхнего и правого нижнего углов окна.
);
```

Если функция не отрабатывает, то возвращается ноль, если функция выполняется успешно, то возвращается отличное от нуля значение.

SetWindowPos - Изменяет размер, положение и Z-порядок дочернего, всплывающего окна или окна верхнего уровня. Эти окна упорядочены в соответствии с их внешним видом на экране. Самое верхнее окно получает наивысший ранг и является первым окном в Z-порядке.

```
BOOL SetWindowPos(
  HWND hWnd, // Дескриптор окна
  HWND hWndInsertAfter, // Дескриптор окна, предшествующего позиционированному окну в Z-порядке. Этот параметр должен быть дескриптором окна или одним из следующих значений: (HWND_BOTTOM, HWND_NOTOPMOST, HWND_TOP, HWND_TOPMOST)
  int X, // Новое положение левой части окна в клиентских координатах.
  int Y, // Новое положение верхней части окна в клиентских координатах.
  int cx, // Новая ширина окна в пикселях.
  int cy, // Новая высота окна в пикселях.
  UINT uFlags // Флаги изменения размера и положения окна. Этот параметр может быть комбинацией значений.
);
```

Расшифровка значений второго параметра (`hWndInsertAfter`):

`HWND_BOTTOM:` помещает окно в нижней части Z-порядка. Если параметр hWnd определяет самое верхнее окно, окно теряет свой самый верхний статус и помещается внизу всех остальных окон.

`HWND_NOTOPMOST:` помещает окно над всеми не самыми верхними окнами (то есть позади всех самых верхних окон). Этот флаг не действует, если окно уже не является верхним окном.

`HWND_TOP:` помещает окно вверху Z-порядка.

`HWND_TOPMOST:` помещает окно над всеми не самыми верхними окнами. Окно сохраняет свое самое верхнее положение, даже когда оно отключено.

Если функция не отрабатывает, то возвращается ноль, если функция выполняется успешно, то возвращается отличное от нуля значение.

MoveWindow - Изменяет положение и размеры указанного окна. Для окна верхнего уровня положение и размеры указаны относительно левого верхнего угла экрана. Для дочернего окна они относятся к левому верхнему углу клиентской области родительского окна.

```
BOOL MoveWindow(
  HWND hWnd, // Дескриптор окна
  int X, // Новое положение левой части окна в клиентских координатах.
  int Y, // Новое положение верхней части окна в клиентских координатах.
  int nWidth, // Новая ширина окна.
  int nHeight, // Новая высота окна.
  BOOL bRepaint // Указывает, нужно ли перекрашивать окно.
);
```

Если параметр bRepaint имеет значение **TRUE**, окно получает сообщение, если же он имеет значение **FALSE**, то никакой перерисовки не происходит. Это относится к клиентской области, неклиентской области (включая строку заголовка и полосы прокрутки) и любой части родительского окна, открывшейся в результате перемещения дочернего окна.

Если функция не отрабатывает, то возвращается ноль, если функция выполняется успешно, то возвращается отличное от нуля значение.

Например:

```
RECT R={100,100,300,200};
MoveWindow(hWnd,R.left+x,R.top+y,R.right-R.left,R.bottom-R.top,true);
```

MessageBox - Отображает модальное диалоговое окно, содержащее значок системы, набор кнопок и краткое сообщение о приложении, например информацию о состоянии или ошибке. Окно сообщения возвращает целочисленное значение, которое указывает, какую кнопку нажал пользователь.

```
int MessageBox(
  HWND    hWnd, // Дескриптор окна владельца создаваемого окна сообщения. Если этот параметр равен NULL, окно сообщения не имеет окна владельца.
  LPCTSTR lpText, // Сообщение, которое будет отображаться. Если строка состоит из более чем одной строки, вы можете разделить строки, используя символ возврата каретки и / или перевода строки между каждой строкой.
  LPCTSTR lpCaption, // Заголовок диалогового окна. Если этот параметр - ПУСТО (NULL), заголовок по умолчанию - Ошибка.
  UINT    uType // Содержание и поведение диалогового окна. Этот параметр может быть комбинацией флагов из групп флагов.
);
```

Чтобы указать кнопки, отображаемые в окне сообщения, нужно указать одно из следующих значений параметра `uType`:

MB_ABORTRETRYIGNORE – три кнопки: «Прервать», «Повторить» и «Игнорировать».

MB_CANCELTRYCONTINUE - три кнопки: «Отмена», «Повторить попытку», «Продолжить».

MB_HELP - Добавляет кнопку справки в окно сообщения. Когда пользователь нажимает кнопку справки или нажимает клавишу F1, система отправляет владельцу сообщение WM_HELP.

MB_OK – Одна кнопка: «ОК». Это значение по умолчанию.

MB_OKCANCEL - две кнопки: «ОК» и «Отмена».

MB_RETRYCANCEL - две кнопки: «Повторить» и «Отмена».

MB_YESNO - две кнопки: «Да» и «Нет».

MB_YESNOCANCEL - три кнопки: «Да», «Нет» и «Отмена».

Возвращаемое значение (соответствует нажатой кнопке):

IDABORT(3), IDCANCEL(2), IDCONTINUE(11), IDIGNORE(5), IDNO(7), IDOK(1), IDRETRY(4), IDTRYAGAIN(10), IDYES(6)

Например:

```
MessageBox(hWnd,"Текст сообщения"," ",MB_OK);
```

CreateWindow - Создает перекрывающееся всплывающее или дочернее окно. Он определяет класс окна, заголовок окна, стиль окна и (необязательно) начальную позицию и размер окна. Функция также определяет родителя или владельца окна, если таковые имеются, и меню окна.

```
HWND CreateWindow (
  LPCTSTR lpClassName, // зарегистрированное имя класса
  LPCTSTR lpWindowName, // имя окна
  DWORD dwStyle, // стиль окна
  int x, // x – координата позиции окна
  int y, // y – координата позиции окна
  int nWidth, // ширина окна
  int nHeight, // высота окна
  HWND hWndParent , // указатель родительского окна
  HMENU hMenu, // дескриптор меню, NULL, если меню нет
  HINSTANCE hInstance, // дескриптор приложения
  LPVOID lParam // данные, которые могут быть переданы для создания окна
  // NULL, если данные не передаются
);
```

Например:

```
HWND hBut1;
hBut1=CreateWindow( TEXT("BUTTON"), TEXT("Вычислить"), WS_VISIBLE | WS_CHILD ,
100, 100, 80, 26, hWnd, NULL, hInstance, NULL) ;
```

Если функция завершается успешно, возвращаемое значение - дескриптор нового окна. Если функция не работает, возвращаемое значение - NULL.

SetWindowLong - Изменяет атрибут указанного окна. Функция также устанавливает 32-битное (длинное) значение с указанным смещением в дополнительную память окна.

```
LONG SetWindowLong(
  HWND hWnd, // Дескриптор окна
  int nIndex, // Значение смещения, которое устанавливается
  LONG dwNewLong //Новое значение
);
```

Параметр `nIndex`**:**

Определяет значение смещения, отсчитываемое от нуля, которое будет установлено. Допустимые значения находятся в диапазоне от нуля до числа байтов дополнительного пространства в памяти, минус 4; например, если бы Вы установили 12 или большее количество байтов памяти дополнительного пространства, значение 8 было бы индексом к третьему 32-разрядному целому числу. Чтобы установить любое другое значение, определите одно из следующих значений:

GWL_EXSTYLE - Устанавливает новый расширенный стиль окна.

GWL_STYLE - Устанавливает новый стиль окна.
GWL_WNDPROC - Устанавливает новый адрес для оконной процедуры.
GWL_HINSTANCE – Устанавливает новый дескриптор экземпляра прикладной программы.
GWL_ID - Устанавливает новый идентификатор окна.
GWL_USERDATA - Устанавливает 32-разрядное значение, связанное с окном. Каждое окно имеет соответствующее 32-разрядное значение, предназначенное для использования прикладной программой, которая создала окно.

Пример:

```
if (hBut1!=NULL)  SetWindowLong(hBut1,GWL_ID,999);
```

Если функция завершается успешно, возвращаемое значение – предыдущее значение указанного 32-битного целого числа.
Если функция не работает, возвращаемое значение - ноль.

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.

```
int GetWindowText(
  HWND hWnd,            // дескриптор окна или элемента управления с текстом
  LPTSTR lpString,              // адрес буфера для текста
  int nMaxCount                 // максимальное число символов для копирования
);
```

Если функция завершается успешно, возвращаемое значение - длина, в символах, скопированной строки, не, включая символа конца строки (ноль-терминатора). Если у окна нет никакой строки заголовка или текста, если строка заголовка пустая или, если дескриптор окна или элемента управления недопустимы, возвращаемое значение нулевое.
Эта функция заставляет послать сообщение WM_GETTEXT определяемому окну или элементу управления. Эта функция не может возвращать текст элемента для редактирования в другую прикладную программу.

SetWindowText изменяет текст строки заголовка указанного окна (если она есть). Если указанное окно является элементом управления, текст элемента управления изменяется. Однако SetWindowText не может изменить текст элемента управления в другом приложении.

```
BOOL SetWindowText(
  HWND hWnd,            // дескриптор окна или элемента управления
  LPCTSTR lpString              // адрес строки
);
```

Если функция завершилась успешно, возвращается значение отличное от нуля. Если функция потерпела неудачу, возвращаемое значение - ноль.
Функция SetWindowText вызывает сообщение WM_SETTEXT, которое будет послано определяемому окну или элементу управления. Если окно - элемент управления окна со списком, созданное в стиле WS_CAPTION, то, несмотря на это, SetWindowText устанавливает текст для элемента управления, а не для ввода в окно списка. Функция SetWindowText не увеличивает размер символов табуляции (код ASCII 0x09). Символы табуляции отображаются как символы вертикальной черты (|).
Пример:

```
char str[50];
GetWindowText(hEd1,str,50);
```

```
SetWindowText(hSt1,str);
```

SetParent изменяет родительское окно указанного дочернего окна.

```
HWND SetParent(
  HWND hWndChild,      // Дескриптор окна, родитель которого изменяется
  HWND hWndNewParent   // Дескриптор нового родительского окна
);
```

Если второй параметр (`hWndNewParent`) - NULL, окно рабочего стола становится новым родительским окном.
Если функция завершается успешно, возвращаемое значение - дескриптор предыдущего родительского окна. Если функция не выполняет задачу, возвращаемое значение - NULL.

GetParent извлекает дескриптор родителя или владельца указанного окна.

```
HWND GetParent(
  HWND hWnd    // дескриптор дочернего окна
);
```

Если окно является дочерним окном, возвращаемое значение - дескриптор родительского окна. Если окно является окном верхнего уровня со стилем `WS_POPUP`, возвращаемое значение - дескриптор окна владельца.
Если функция не работает, возвращаемое значение - NULL.

Эта функция обычно не работает по одной из следующих причин:

1. Окно — это окно верхнего уровня, которое не имеет владельца или не имеет стиля `WS_POPUP`.
2. Окно владельца имеет стиль `WS_POPUP`.

Sleep приостанавливает выполнение текущего потока до истечения времени ожидания.

```
void Sleep(
  DWORD dwMilliseconds // Интервал времени, на который должно быть
приостановлено выполнение, в миллисекундах.
);
```

Функции для работы с регионами:

CombineRgn - объединяет две области и сохраняет результат в третьей. Две области объединяются согласно указанному режиму объединения.

```
int WINAPI CombineRgn(
  HRGN hrgnDest, // дескриптор новой области
  HRGN hrgnSrc1, // первая исходная область
  HRGN hrgnSrc2, // вторая исходная область
  int fnCombineMode // режим комбинирования
);
```

Способ комбинирования областей зависит от значения параметра `fnCombineMode`:
`RGN_AND` - Пересечение областей `hrgnSrc1` и `hrgnSrc2`
`RGN_OR` - Объединение областей `hrgnSrc1` и `hrgnSrc2`
`RGN_XOR` - Объединение областей `hrgnSrc1` и `hrgnSrc2` с исключением перекрывающихся областей

RGN_DIFF - Область hrgnSrc1, которая не входит в область hrgnSrc2
RGN_COPY - Область hrgnSrc1

CombineRegion может вернуть одно из следующих значений:
ERROR – Ошибка
NULLREGION - Новая область пустая
SIMPLEREGION - Новая область не является самопересекающейся (т. е. граница созданной области не пересекает саму себя)
COMPLEXREGION - Создана самопересекающаяся область

CreateEllipticRgn создает эллиптическую область.

```
HRGN CreateEllipticRgn(
  int nLeftRect, // x-координата верхнего левого угла ограничивающего прямоугольника
  int nTopRect, // y-координата верхнего левого угла ограничивающего прямоугольника
  int nRightRect, // x-координата нижнего правого угла ограничивающего прямоугольника
  int nBottomRect // y-координата нижнего правого угла ограничивающего прямоугольника
);
```

Если функция завершается успешно, возвращаемое значение - дескриптор региона.
Если функция не работает, возвращаемое значение - NULL.

CreatePolygonRgn создает многоугольную область.

```
HRGN CreatePolygonRgn (
  CONST POINT * lppt,  // указатель на массив точек
  int cPoints,         // число точек в массиве
  int fnPolyFillMode   // режим заполнения многоугольника
);
```

Если функция завершается успешно, возвращаемое значение - дескриптор региона.
Если функция не работает, возвращаемое значение - NULL.

CreateRectRgn создает прямоугольную область.

```
HRGN CreateRectRgn (
  int nLeftRect,       // x-координата верхнего левого угла области
  int nTopRect,        // y-координата верхнего левого угла области
  int nRightRect,      // x-координата правого нижнего угла области
  int nBottomRect      // y-координата правого нижнего угла области
);
```

Если функция завершается успешно, возвращаемое значение - дескриптор региона.
Если функция не работает, возвращаемое значение - NULL.

CreateRoundRectRgn создает прямоугольную область с закругленными углами.

```
HRGN CreateRoundRectRgn (
  int nLeftRect,       // x -координата верхнего левого угла области
  int nTopRect,        // y -координата верхнего левого угла области
  int nRightRect,      // x -координата нижнего правого угла области
```

```
  int nBottomRect,     // y -координата нижнего правого угла области
  int nWidthEllipse,   // ширина эллипса для закругленных углов
  int nHeightEllipse   // высота эллипса для закругленных углов
);
```

Если функция завершается успешно, возвращаемое значение - дескриптор региона.
Если функция не работает, возвращаемое значение - NULL.

SetWindowRgn - Прикрепляет регион к указанному окну. Регион окна устанавливает область внутри окна, где система разрешает рисовать. Система не выводит на экран любую часть окна, которая находится за пределами региона окна.

```
int SetWindowRgn(
  HWND hWnd,    // дескриптор окна
  HRGN hRgn,    // дескриптор региона
  BOOL bRedraw // опции перерисовки окна (перерисовывать или нет)
);
```

Если функция завершается успешно, возвращаемое значение отлично от нуля.
Если функция завершается ошибкой, возвращаемое значение - нуль.

Пример: задание региона и окна в форме эллипса:

```
HRGN FormRgn;
GetWindowRect(hWnd,&WRect);
FormRgn = CreateEllipticRgn(0,0,WRect.right-WRect.left,WRect.bottom-WRect.top);
SetWindowRgn(hWnd,FormRgn,true);
```

Описание событий мыши с параметрами:

`WM_LBUTTONDBLCLK` - посылается тогда, если пользователь дважды щелкает левой кнопкой 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 посылается в окно, которое захватило мышь.
`WM_LBUTTONDOWN` посылается, если пользователь нажимает левую кнопку мыши
`WM_LBUTTONUP` посылается, если пользователь отпускает левую кнопку мыши
`WM_MBUTTONDBLCLK` посылается тогда, когда пользователь щелкает два раза средней кнопкой мыши
`WM_MBUTTONDOWN` посылается тогда, когда пользователь нажимает среднюю кнопку мыши
`WM_MBUTTONUP` посылается тогда, когда пользователь отпускает среднюю кнопку мыши
`WM_MOUSEACTIVATE` отправляется тогда, когда курсор находится в неактивном окне, а пользователь нажимает кнопку мыши. Родительское окно получает это сообщение, только в том случае, если дочернее окно передает его в функцию `DefWindowProc`.
`WM_MOUSEHOVER` посылается в окно, когда курсор нависает над рабочей областью окна в течение периода времени, определенного при предшествующем вызове функции `TrackMouseEvent`
`WM_MOUSELEAVE` посылается в окно тогда, когда курсор оставляет рабочую область окна, заданную при предшествующем вызове функции `TrackMouseEvent`.
`WM_MOUSEWHEEL` отправляется в окно с фокусом, когда прокручивается колесико мыши. Функция `DefWindowProc` передает сообщение родителю окна. Не должно быть никакой внутренней переадресации сообщения, так как `DefWindowProc` передает его вверх по цепочке родителей, пока функция не найдет окно, которое обработает его.
`WM_RBUTTONDBLCLK` посылается тогда, когда пользователь дважды щелкает правой кнопкой мыши
`WM_RBUTTONDOWN` посылается, если пользователь нажимает правую кнопку мыши
`WM_RBUTTONUP` посылается тогда, когда пользователь отпускает правую кнопку мыши

Параметры событий мыши:

wParam

Указывает, находятся ли в нажатом состоянии различные виртуальные клавиши. Этот параметр может состоять из одного или нескольких нижеперечисленных значений.

MK_CONTROL	Клавиша CTRL находится в нажатом состоянии.
MK_LBUTTON	Левая кнопка мыши находится в нажатом состоянии.
MK_MBUTTON	Средняя кнопка мыши находится в нажатом состоянии.
MK_RBUTTON	Правая кнопка мыши находится в нажатом состоянии.
MK_SHIFT	Клавиша SHIFT находится в нажатом состоянии.

lParam

Младшее слово устанавливает x-координату курсора. Координата - относительно левого верхнего угла рабочей области.
Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области.

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

События WM_CREATE, WM_COMMAND, WM_DESTROY и WM_QUIT.

WM_CREATE 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.

Элемент управления может передать в процедуру диалогового окна сообщение **WM_COMMAND**, когда пользователь завершает деятельность в элементе управления. Эти сообщения, называемые уведомительными сообщениями, информируют процедуру, что пользователь ввел данные и дал разрешение на выполнение присущей ей работы.

WM_DESTROY 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_QUIT указывает запрос, чтобы завершить прикладную программу и создается, когда из прикладной программы вызвана функция `PostQuitMessage`. Это вынуждает функцию `GetMessage` возвратить нуль.

Типы в Win32:

Тип HANDLE обозначает 32-разрядное целое, используемое в качестве дескриптора. Есть несколько похожих типов данных, но все они имеют ту же длину, что и HANDLE, и начинаются с литеры H. Дескриптор — это просто число, определяющее некоторый ресурс.
Тип HWND обозначает 32-разрядное целое — дескриптор окна.
Тип BYTE обозначает 8-разрядное беззнаковое символьное значение.
Тип WORD — 16-разрядное беззнаковое короткое целое.
Тип DWORD — беззнаковое длинное целое.
Тип UINT — беззнаковое 32-разрядное целое.
Тип LONG эквивалентен типу long.
Тип BOOL обозначает целое и используется, когда значение может быть либо истинным, либо ложным.
Тип LPSTR определяет указатель на строку.

Тип LPCSTR определяет константный (const) указатель на строку.

Ход выполнения работы

1. Создать приложение Win32 Project (в Microsoft Visual Studio).
2. Работа с ресурсами приложения:
   2.1. Изменить заголовок окна (в заголовке указать номер лабораторной работы и фамилию студента); - изменены через файл ресурсов (строка IDS_APP_TITLE)
   2.2. Изменить иконку приложения (создать собственную);
   Добавим иконку в папку проекта, импортируем ее в ресурсы с именем `IDI_LAB01ICON`,
   Изменим поля в структуре класса окна:
   ```
   wcex.hIcon   = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB01ICON));
   wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_LAB01ICON));
   ```
   Это заменит иконку приложения в проводнике и в углу окна.
   2.3. Добавить в меню новый пункт.
   Откроем редактор меню через окно ресурсов, добавим вкладку вид и в ней пункт меню «Сменить родителя кнопки», а также изменим свойство ИД на IDM_CHANGEPARENT (подготовка для выполнения пункта 6)
3. Задать новые координаты (левый верхний угол, например, (100, 100)) и размеры запускаемого приложения (например, 300 и 200). Продумать несколько возможных вариантов изменения.
   Вариант 1: функция `SetWindowPos`
   ```
   SetWindowPos(hWnd, HWND_TOPMOST, 100, 100, 300, 200, SWP_SHOWWINDOW);
   ```
   Вариант 2: функция `MoveWindow`
   ```
   MoveWindow(hWnd, 100, 100, mWindowWidth, mWindowHeight, true);
   ```
4. Создать управляющие элементы пользовательского интерфейса: две кнопки (BUTTON), поле ввода (EDIT) и поле вывода (STATIC). Для кнопок задать идентификатор управляющего элемента.
   Глобальные переменные для этих элементов:
   ```
   // пункт 4 лабораторной работы
   HWND btnLeft;       // кнопка слева (у нее меняем родителя в 6)
   HWND btnRight;      // кнопка справа
   HWND editText;      // поле ввода (EDIT)
   HWND staticText;    // поле вывода (STATIC)

   const static int idBtnLeft = 1;
   const static int idBtnRight = 2;
   const static int idEditText = 3;
   const static int idStaticText = 4;
   ```

   Создаем окна этих элементов в `InitInstance`

```
      btnLeft = CreateWindow(L"BUTTON", L"Left button", WS_VISIBLE | WS_CHILD |
BS_PUSHBUTTON , 25, 150, 100, 25, hWnd, (HMENU)idBtnLeft, hInstance, NULL);
      btnRight = CreateWindow(L"BUTTON", L"Right button", WS_VISIBLE | WS_CHILD |
BS_PUSHBUTTON, 150, 150, 100, 25, hWnd, (HMENU)idBtnRight, hInstance, NULL);
      editText = CreateWindow(L"EDIT", L"I am EDIT", WS_VISIBLE | WS_CHILD | WS_BORDER, 25,
25, 225, 25, hWnd, (HMENU)idEditText, hInstance, NULL);
      staticText = CreateWindow(L"STATIC", L"I am STATIC", WS_VISIBLE | WS_CHILD, 25, 75,
225, 25, hWnd, (HMENU)idStaticText, hInstance, NULL);
```

5. Написать обработчики событий для кнопок, пункта меню и некоторых событий:
   5.1. 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;

На события WM_CREATE, WM_DESTROY будем выводить MessageBox, а также код завершения программы на WM_DESTROY (`PostQuitMessage(0);`)

На нажатие левой и правой кнопок мыши будем выводить соответствующий текст в STATIC (`staticText`):

```
case WM_LBUTTONDOWN:
 {
       SetWindowText(staticText, L"Left mouse button pressed");
 }
 break;
case WM_RBUTTONDOWN:
 {
       SetWindowText(staticText, L"Right mouse button pressed");
 }
 break;
```

На нажатие на меню (`WM_MENUSELECT` или `WM_INITMENUPOPUP`) будем также выводить сообщение:

```
case WM_INITMENUPOPUP:
 {
     SetWindowText(staticText, L"Menu command!");
 }
 break;
```

5.2. Первая кнопка BUTTON: текст введенный в поле EDIT должен появиться в поле STATIC;

Внутри обработчика сообщений от управляющих элементов (`WM_COMMAND`) напишем реакцию на значение параметра `wmId`

```
case idBtnLeft: // нажатие на левую кнопку - вставляем в STATIC текст из EDIT
     {
         TCHAR editTextContent[MAX_LOADSTRING];
         GetWindowText(editText, editTextContent, MAX_LOADSTRING);
         SetWindowText(staticText, editTextContent);
     }
     break;
```

5.3. Вторая кнопка BUTTON: в поле STATIC должен быть выведен заголовок окна приложения.

```
case idBtnRight: // нажатие на правую кнопку - вставляем в STATIC заголовок окна
     {
          TCHAR editTextContent[MAX_LOADSTRING];
          GetWindowText(hWnd, editTextContent, MAX_LOADSTRING);
          SetWindowText(staticText, editTextContent);
     }
     break;
```

6. Обработчик пункта меню должен: сменить окно - родителя у первой кнопки, а затем заставить ее медленно перемещаться по экрану. Нажатие правой кнопки мыши должно обеспечить возвращение кнопки в окно программы.
7.

Глобальные переменные для таймера:

```
const static int idTimer = 5; // идентификатор таймера
static int timerTick = 0; // счетчик тиков
```

Обработчик нажатия пункта меню:

```
case IDM_CHANGEPARENT:
{
	timerTick = 0; // обнулить таймер
	SetTimer(hWnd, idTimer, 50, (TIMERPROC)NULL); // таймер с периодом 50мс
	if (GetParent(btnLeft) == hWnd) {   // если в главном окне
		SetParent(btnLeft, NULL);
	}
	else { // где-то еще
		KillTimer(hWnd, idTimer); // остановить таймер
		SetParent(btnLeft, hWnd); // устанавливаем главное окно как родителя
		MoveWindow(btnLeft, 25, 150, 100, 25, true); // перемещаем на старое
	место
	}
}
```

Обработчик тика таймера:

```
case WM_TIMER:
{
	++timerTick;
	MoveWindow(btnLeft, timerTick * 2, timerTick, 100, 25, true);
}
break;
```

Допишем обработчик нажатия на правую кнопку мыши:

```
case WM_RBUTTONDOWN: // Нажатие на правую кнопку мыши
{
	SetWindowText(staticText, L"Right mouse button pressed");
	KillTimer(hWnd, idTimer);
	if (GetParent(btnLeft) != hWnd) {
		SetParent(btnLeft, hWnd);
		MoveWindow(btnLeft, 25, 150, 100, 25, true);
	}
}
break;
```

8. Оформить окно приложения в форме региона: прямоугольник с эллипсом (нижний край окна) и вырезанным посередине кругом (расположить все управляющие элементы, чтобы они были видны). Проверить работоспособность приложения.

Оформим функцию для изменения региона окна:

```
void mWindowRecombineRgn(HWND hWnd) {
    HRGN cutRegion, outRegion;
    RECT mWindowRect;
    GetWindowRect(hWnd, &mWindowRect);
    mWindowWidth = mWindowRect.right - mWindowRect.left; // новая ширина окна
    mWindowHeight = mWindowRect.bottom - mWindowRect.top; // новая высота окна
    outRegion = CreateEllipticRgn(0, -mWindowHeight, mWindowWidth, mWindowHeight);
    int r = 30; // радиус вырезанного круга
    cutRegion = CreateEllipticRgn(mWindowWidth - 100 - r, 140 - r, mWindowWidth - 100
+ r, 140 + r);
    CombineRgn(cutRegion, cutRegion, outRegion, RGN_XOR);
    SetWindowRgn(hWnd, cutRegion, true);
}
```

Вызовем ее после перемещения окна:

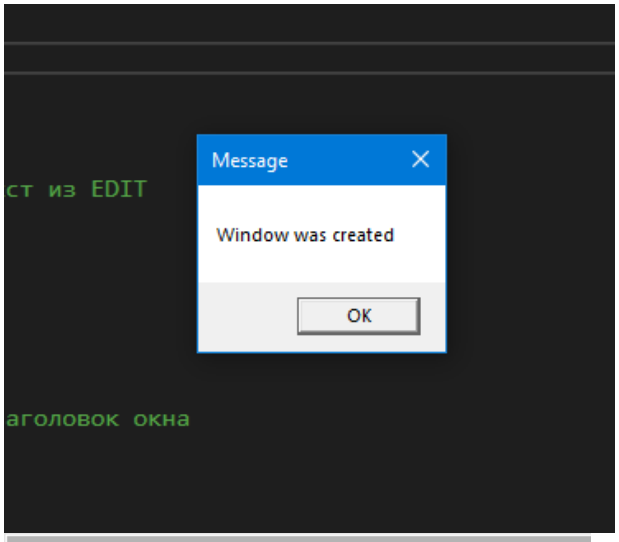
```
mWindowRecombineRgn(hWnd);
```

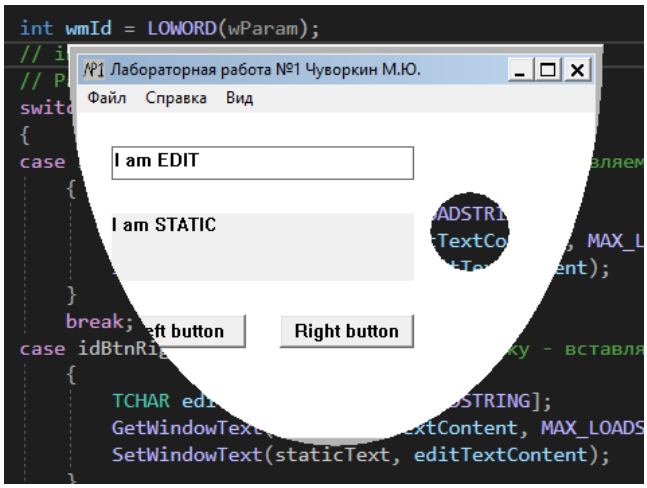
А также будем вызывать ее, если меняем размер окна:

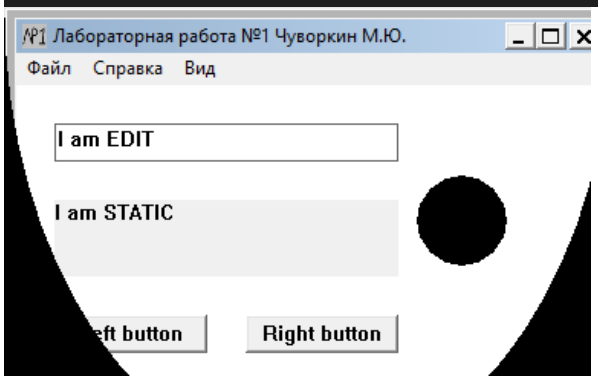
```
        case WM_SIZE:
         {
             mWindowRecombineRgn(hWnd);
         }
         break;
```

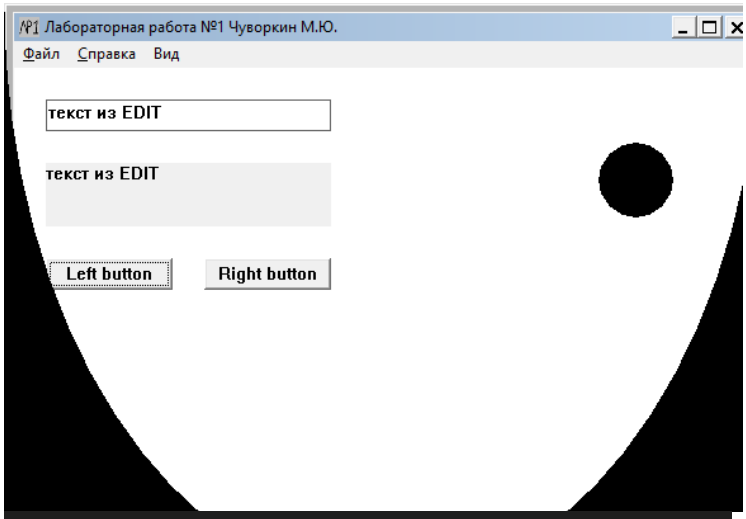
Полный код программы доступен здесь: https://pastebin.com/0BGrrbdx (в Word код нечитаемым получается)

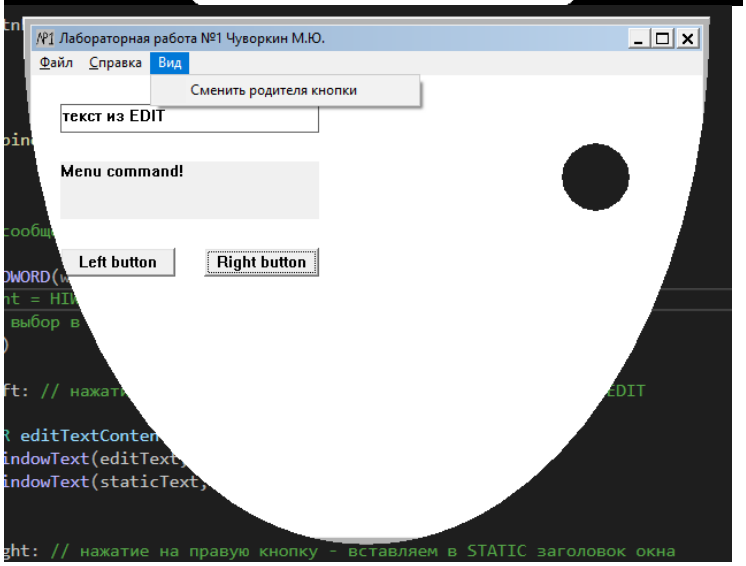
Проверка работоспособности программы:

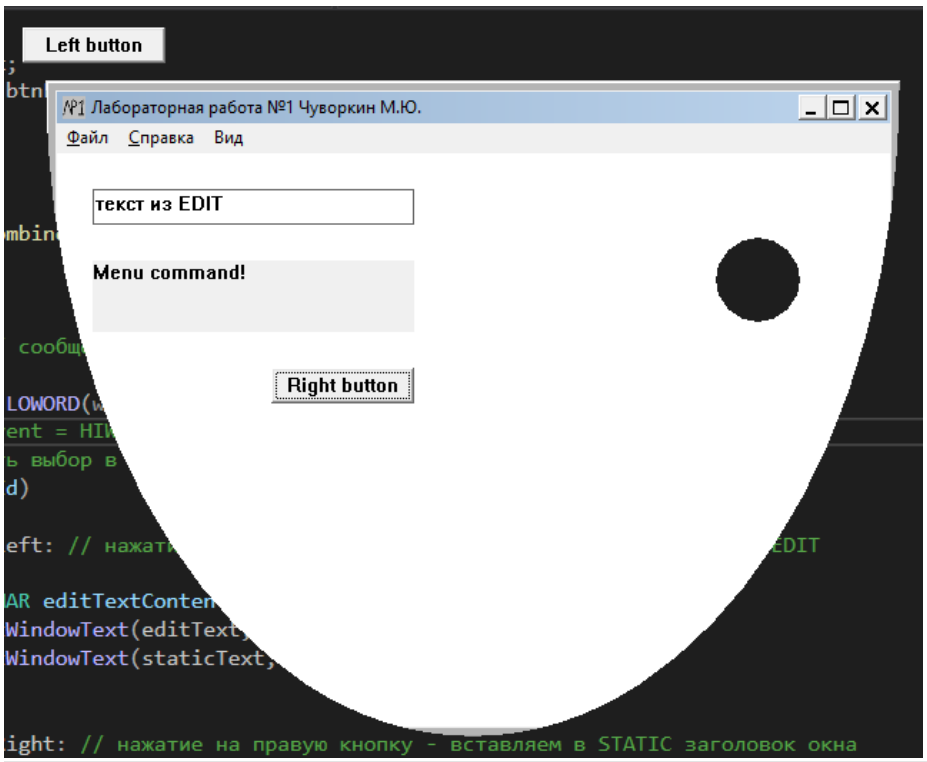


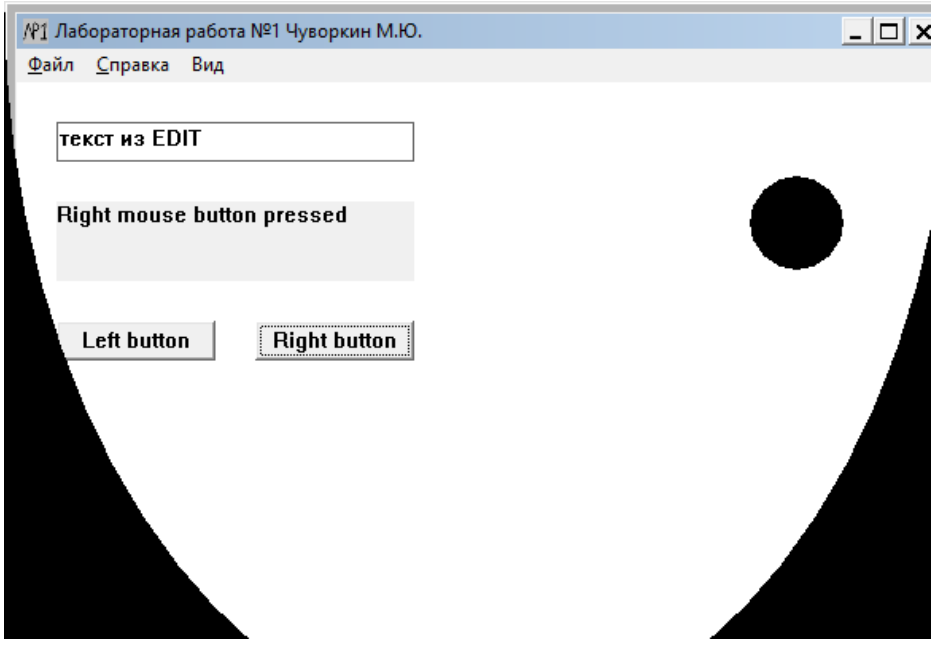


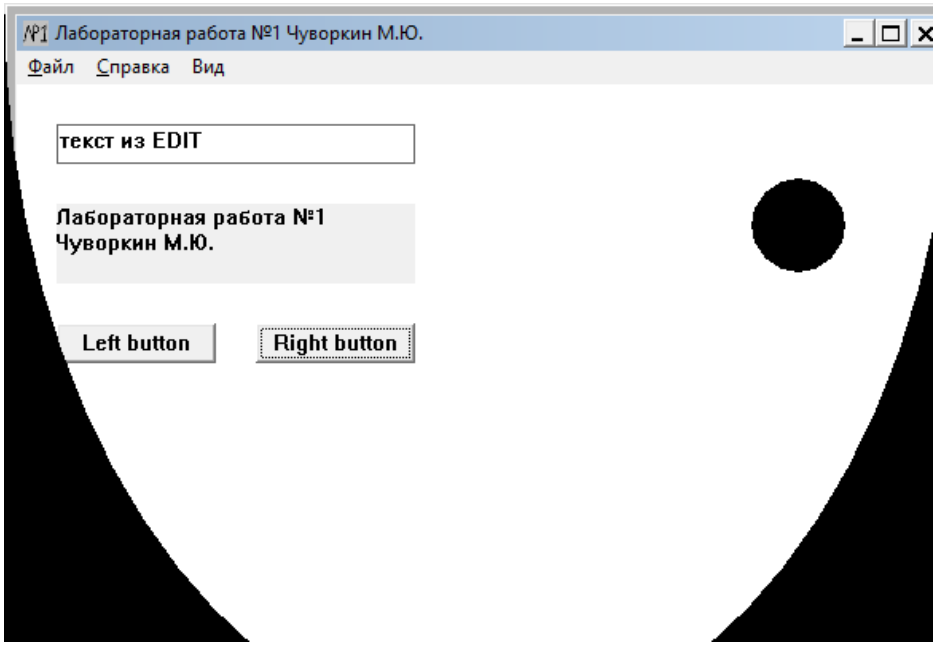


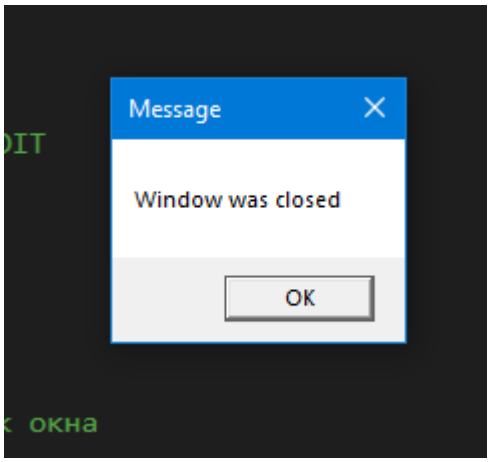

Лабораторная работа №1 Чуворкин М.Ю.
Файл Справка Вид
текст из EDIT
текст из EDIT
Left button
Right button


Лабораторная работа №1 Чуворкин М.Ю.
Файл Справка Вид
Сменить родителя кнопки
текст из EDIT
Menu command!
Left button
Right button


Left button
№1 Лабораторная работа №1 Чуворкин М.Ю.
Файл Справка Вид
текст из EDIT
Menu command!
Right button
LOWORD(
ent = HI
ь выбор в
d)
eft: // нажат
EDIT
AR editTextConte
WindowText(editText
WindowText(staticText
ight: // нажатие на правую кнопку - вставляем в STATIC заголовок окна


№1 Лабораторная работа №1 Чуворкин М.Ю.
Файл Справка Вид
текст из EDIT
Right mouse button pressed
Left button
Right button


Лабораторная работа №1 Чуворкин М.Ю.
Файл Справка Вид
текст из EDIT
Лабораторная работа №1
Чуворкин М.Ю.
Left button
Right button


Message
Window was closed
OK