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

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

|  |  |
| --- | --- |
|  |  |
| **Предмет: «Системное программирование»**  **Отчет по лабораторной работе №1**  **Выполнил: студент группы А-13а-19**  **Рамазанов Н. М.**  **Преподаватель: Меньшикова К. Г.**  **Москва, 2021** | |

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

1) Получение координат окна GetWindowRect.

BOOL GetWindowRect (HWND hWnd, LPRECT lpRect);

Параметры: hWnd – дескриптор окна для чтения позиции и размера окна; lpRect - переменная, которая получает координаты левого верхнего и правого нижнего углов окна.

В случае ошибки функция возвращает 0. В успешном случае возвращается ненулевое значение.

2) Установление координат окна SetWindowPos.

BOOL SetWindowPos ( HWND hWnd, HWND hWndInsertAfter, int x,

int y, int cx, int cy, UINT uFlags);

Параметры: hWnd – дескриптор окна для чтения позиции и размера окна; hWndInsertAfter – дескриптор окна, после которого окно hWnd находится в списке окон; x, y – новые координаты левого верхнего угла окна; cx – новая ширина окна; cy – новая высота окна; uFlags – флаги, определяющие вид окна.

В случае ошибки функция возвращает 0. В успешном случае возвращается ненулевое значение.

3) Перемещение окна MoveWindow.

BOOL MoveWindow ( HWND hWnd, int x, int y, int nWidth,

int nHeight, BOOL bRepaint);

Параметры: hWnd – дескриптор окна для чтения позиции и размера окна; x, y – новые координаты левого верхнего угла окна; nWidth – новая ширина окна; nHeight – новая высота окна; bRepaint\* – определяет, должно ли окно быть перерисовано.

\* Если этот параметр ненулевой, окно принимает сообщение WM\_PAINT. Если 0, то никакого перекрашивания какого-либо сорта не происходит.

В случае ошибки функция возвращает 0. В успешном случае возвращается ненулевое значение.

4) Вывод сообщения MessageBox.

int MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption,

UINT uType);

Параметры: hWnd – дескриптор родительского окна (если NULL, окно создается без родителя); lpText - текст, выводимый в окне сообщения; lpCaption - текст, выводимый в заголовке; uType\* - устанавливает содержание и режим работы диалогового окна.

\*Данный параметр принимает следующие значения (перечислены основные):

*MB\_ABORTRETRYIGNORE= &H2*

Окно сообщения содержит три кнопки: Стоп, Повтор и Пропустить

*MB\_CANCELTRYCONTINUE=&H6*

Окно сообщения содержит три кнопки: Cancel, Try Again и Continue

*MB\_HELP=&H4000*

Добавляется кнопка Справка, при нажатии которой посылается сообщение WM\_HELP родительскому окну (определяется параметром hWnd). Данный флаг может использоваться только вместе с другими флагами

*MB\_OK=&H0*

Окно сообщения содержит только одну кнопку: OK. Используется по умолчанию

*MB\_OKCANCEL=&H1*

Окно сообщения содержит две кнопки: OK и Отмена

*MB\_RETRYCANCEL=&H5*

Окно сообщения содержит две кнопки: Повтор и Отмена

*MB\_YESNO=&H4*

Окно сообщения содержит две кнопки: Да и Нет

*MB\_YESNOCANCEL=&H3*

Окно сообщения содержит три кнопки: Да, Нет и Отмена

В случае ошибки функция возвращает 0. В успешном случае функция возвращает один из следующих флагов, определяющих нажатую кнопку:

*IDABORT=3* Нажата кнопка Стоп

*IDCANCEL=2* Нажата кнопка Отмена (или Esc)

*IDCONTINUE=11* Нажата кнопка Continue

*IDIGNORE=5* Нажата кнопка Пропустить

*IDNO=7* Нажата кнопка Нет

*IDOK=1* Нажата кнопка OK

*IDRETRY=4* Нажата кнопка Повтор

*IDTRYAGAIN=10* Нажата кнопка Try Again

*IDYES=6* Нажата кнопка Да

5) Создание окна CreateWindow.

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

Параметры: lpClassName **−** указатель на зарегистрированное имя класса; lpWindowName **−** указатель на имя (текст) окна; dwStyle **−** стиль окна; x, y **−** горизонтальная и вертикальная позиция окна; nWidth*,* nHeight **−** ширина и высота окна; hWndParent **−** дескриптор родительского или окна владельца; hMenu **−** дескриптор меню или идентификатор дочернегоокна; hInstance **−** дескриптор экземпляра приложения; lpParam **−** указатель на данные создания окна.

В случае ошибки функция возвращает NULL. В успешном случае возвращается дескриптор созданного окна.

6) Установка идентификатора SetWindowLong.

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

Параметры: hWnd – дескриптор окна и, косвенно, класс, к которому принадлежит окно; nIndex - определяет смещение в памяти (следовательно, устанавливаемый атрибут стиля окна); dwNewLong - устанавливает новое значение, определяющее стиль окна.

В случае ошибки функция возвращает 0. В успешном случае возвращается предыдущее значение установленного стиля.

7) Чтение GetWindowText и изменение SetWindowText текста данного окна.

int GetWindowText (HWND hwnd, LPTSTR pszBuffer,

int iMaxLength)

BOOL SetWindowText (HWND hwnd, LPCTSTR pszString);

Параметры: hwnd – дескриптор окна, в котором изменяется текст; pszString – указатель на оканчивающуюся нулем строку; iMaxLength – максимальное число символов для копирования в буфер, который определяется указателем pszBuffer.

Обе функции в случае ошибки возвращают 0. В успешном случае SetWindowText возвращает ненулевое значение, GetWindowText – длину скопированной строки.

8) Получение GetParent и замена SetParent родительского окна.

HWND GetParent (HWND hWnd);

HWND SetParent (HWND hWndChild, HWND hWndNewParent);

Параметры: hWnd – дескриптор окна, родительское окно которого ищем; hWndChild – дескриптор дочернего окна; hWndNewParent – дескриптор нового родительского окна.

Обе функции в случае ошибки возвращают 0. В успешном случае GetParent возвращает дескриптор родительского окна, если окно – дочернее окно; NULL, если окно – не имеющее владельца окно верхнего уровня; SetParent возвращает дескриптор дочернего окна бывшего родительского окна.

9) Приостановка выполнения программы Sleep.

VOID Sleep (DWORD dwMilliseconds);

Параметр: dwMilliseconds – время (в мс), на которое приостанавливается работы программы.

Функция не возвращает никакого значения.

10) Функции для работы с регионами:

– CombineRgn комбинирует два региона в другой регион различными способами.

int CombineRgn (HRGN hrgnDest, HRGN hrgnSrc1, HRGN hrgnSrc2,

int fnCombineMode);

Параметры: hrgnDest – дескриптор результирующей области; hrgnSrc1, hrgnSrc2 – дескрипторы исходных областей; fnCombineMode – режим объединения областей.

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 не является самопересекающимся

COMPLEXREGION = 3 | Полученный регион является самопересекающимся

– CreateEllipticRgn создает регион в виде эллипса, вписанного в прямоугольник, определенный координатами, передаваемые функцией.

HRGN CreateEllipticRgn (int x1, int y1, int x2, int y2);

Параметры: x1, y1 – координаты верхнего левого угла ограничительного прямоугольника; x2, y2 – координаты его нижнего правого угла.

В случае ошибки функция возвращает 0. В успешном случае возвращается дескриптор созданной области.

– CreatePolygonRgn создает регион в виде многоугольника.

HRGN CreatePolygonRgn (const POINT \*pptl, int cPoint, int iMode);

Параметры: \*pptl – указатель на массив точек, определяющих вершины многоугольника; cPoint – количество точек в массиве; iMode – режим заливки многоугольника.

В случае ошибки функция возвращает 0. В успешном случае возвращается дескриптор созданной области.

– CreateRectRgn создает прямоугольный регион.

HRGN CreateRectRgn (int x1, int y1, int x2, int y2);

Параметры: x1, y1 – координаты верхнего левого угла прямоугольника; x2, y2 – координаты его нижнего правого угла.

В случае ошибки функция возвращает 0. В успешном случае возвращается дескриптор созданной области.

– CreateRoundRectRgn создает прямоугольный регион со скругленными краями.

HRGN CreateRoundRectRgn (int x1, int y1, int x2, int y2, int w, int h);

Параметры: x1, y1 – координаты верхнего левого угла прямоугольника; x2, y2 – координаты его нижнего правого угла, w – ширина эллипса, используемого для создания скругленных углов; h – высота эллипса.

В случае ошибки функция возвращает 0. В успешном случае возвращается дескриптор созданной области.

– SetWindowRgn прикрепляет регион к указанному окну.

int SetWindowRgn(HWND hWnd, HRGN hRgn, BOOL bRedraw);

Параметры: hWnd – дескриптор окна, для которого задается регион; hRgn – дескриптор региона (если равен 0, то показывается целое окно); bRedraw – определяет, надо ли немедленно перерисовывать окно, используя новый регион (True - надо, в противном случае – False).

В случае ошибки функция возвращает 0. В успешном случае возвращается ненулевое значение.

11) Описание событий мыши с параметрами.

Ко всем событиям: если мышь не захвачена, сообщение посылается в окно под курсором. В противном случае, сообщение посылается в окно, которое захватило мышь.

Левая кнопка: WM\_LBUTTONDBLCLK – двойной щелчок; WM\_LBUTTONDOWN – нажатие; WM\_LBUTTONUP – отжатие.

Средняя кнопка: WM\_MBUTTONDBLCLK – двойной щелчок; WM\_MBUTTONDOWN – нажатие; WM\_MBUTTONUP – отжатие.

Правая кнопка: WM\_RBUTTONDBLCLK – двойной щелчок; WM\_RBUTTONDOWN – нажатие; WM\_RBUTTONUP – отжатие.

Параметры:

wParam содержит флажки\*, которые указывают состояние остальных кнопок мыши и клавиш **CTRL** и **SHIFT** во время события мыши.

lParam указывает позицию острия курсора. Младшее слово указывает x-координату острия, а старшее слово указывает y-координату. Все точки на экране определяются относительно левого верхнего угла рабочей области.

\*Параметром wParam может быть комбинация следующих значений:

MK\_CONTROL Нажата клавиша CTRL.

MK\_LBUTTON Нажата левая кнопка мыши.

MK\_MBUTTON Нажата средняя кнопка мыши.

MK\_RBUTTON Нажата правая кнопка мыши.

MK\_SHIFT Нажата клавиша SHIFT.

Дополнительно:

WM\_CREATE – отправляется тогда, когда программа запрашивает, какое окно будет создаваться вызовом функции 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\_COMMAND – отправляется тогда, когда пользователь выбирает командный пункт из меню, когда орган управления отправляет уведомительное сообщение своему родительскому окну, или когда транслируется нажатие клавиши - ускорителя.

WM\_DESTROY - отправляется тогда, когда окно разрушается. Оно отправляется оконной процедуре разрушаемого окна после того, как окно удаляется с экрана.

WM\_QUIT - указывает запрос на завершение работы приложения и создается, когда из прикладной программы вызывается функция PostQuitMessage. Это вынуждает функцию GetMessage возвратить нуль.

**2. Выполнение работы.**

1. Создал приложение Win32 Project (в Microsoft Visual Studio).
2. Работа с ресурсами приложения:
   1. Изменил заголовок окна (в заголовке указал номер лабораторной работы и свою фамилию) – через Окно ресурсов -> String Table -> IDS\_APP\_TITLE
   2. Изменил иконку приложения (загрузил собственную) – через Окно ресурсов -> Добавить -> Ресурс… -> Icon, а затем изменил ID иконки через Окно ресурсов -> Icon и изменил его в программе:

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_MYICON));

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_MYICON));

* 1. Добавил в меню новый пункт – Действия, и добавил подпункт «Сменить родителя», поменял через Окно ресурсов его ID на IDM\_CHANGEPARENT для удобства.

1. Задал новые координаты (левый верхний угол (200, 200)) и размеры запускаемого приложения (800 и 600). Продумал 2 возможных варианта изменения:

SetWindowPos(hWnd, HWND\_TOPMOST, 200, 200, WWidth, WHeight, SWP\_SHOWWINDOW);

MoveWindow(hWnd, 200, 200, WWidth, WHeight, true);

1. Создал управляющие элементы пользовательского интерфейса: две кнопки (BUTTON), поле ввода (EDIT) и поле вывода (STATIC). Для кнопок задал идентификатор управляющего элемента.

Объявление глобальных переменных для этих элементов:

HWND Btn1; // большая кнопка

HWND Btn2; // малая кнопка

HWND EditTxt; // поле ввода (EDIT)

HWND StaticTxt; // поле вывода (STATIC)

const static int idBtn1 = 1; // идентификаторы кнопок и полей

const static int idBtn2 = 2;

const static int idEditTxt = 3;

const static int idStaticTxt = 4;

Непосредственно создание этих элементов:

Btn1 = CreateWindow(L"BUTTON", L"Вывести текст из EDIT в STATIC", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 250, 150, 300, 50, hWnd, (HMENU)idBtn1, hInstance, NULL);

Btn2 = CreateWindow(L"BUTTON", L"Вывести заголовок в STATIC", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 280, 100, 240, 25, hWnd, (HMENU)idBtn2, hInstance, NULL);

EditTxt = CreateWindow(L"EDIT", L"Some text in EDIT", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 25, 25, 200, 50, hWnd, (HMENU)idEditTxt, hInstance, NULL);

StaticTxt = CreateWindow(L"STATIC", L"Some text in STATIC", WS\_VISIBLE | WS\_CHILD, 575, 25, 200, 50, hWnd, (HMENU)idStaticTxt, hInstance, NULL);

1. Написал обработчики событий для кнопок, пункта меню и некоторых событий:
   1. События: WM\_CREATE, WM\_DESTROY, нажатие левой кнопки мыши, нажатие правой кнопки мыши, щелчок на добавленном пункте меню выводят сообщения о том, какое именно событие произошло;

case WM\_CREATE:

MessageBox(hWnd, L"Окно было успешно создано!", L"Message", MB\_OK);

break;  
case WM\_DESTROY:

MessageBox(hWnd, L"Окно было успешно закрыто!", L"Message", MB\_OK);

PostQuitMessage(0);

break;  
case WM\_LBUTTONDOWN: // Нажатие на левую кнопку мыши

SetWindowText(StaticTxt, L"Нажатие левой кнопки мыши!");

break;

case WM\_RBUTTONDOWN: // Нажатие на правую кнопку мыши

SetWindowText(StaticTxt, L"Нажатие правой кнопки мыши!");

break;

case WM\_INITMENUPOPUP: // Сообщение перед раскрытием меню

SetWindowText(StaticTxt, L"Открыто меню!");

break;

* 1. Первая кнопка BUTTON: текст, введенный в поле EDIT, появляется в поле STATIC;

case idBtn1:// нажатие на большую кнопку - вставляем в STATIC текст из EDIT

{

TCHAR editTextContent[MAX\_LOADSTRING];

GetWindowText(EditTxt, editTextContent, MAX\_LOADSTRING);

SetWindowText(StaticTxt, editTextContent);

}

break;

* 1. Вторая кнопка BUTTON: в поле STATIC выводится заголовок окна приложения.

case idBtn2: // нажатие на малую кнопку - вставляем в STATIC заголовок окна

{

TCHAR editTextContent[MAX\_LOADSTRING];

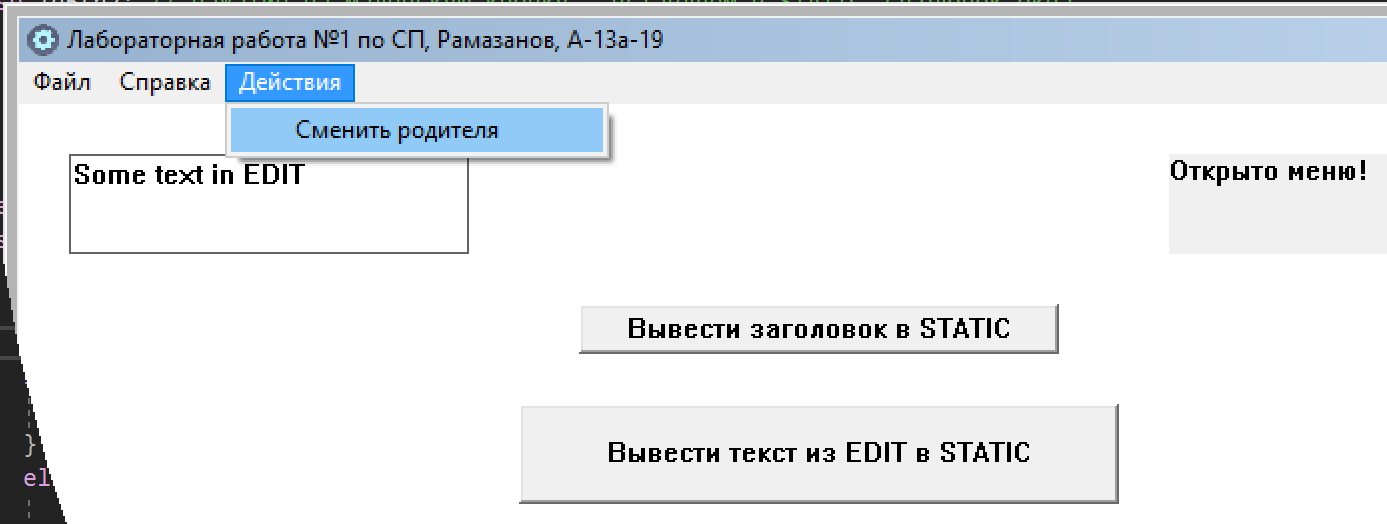
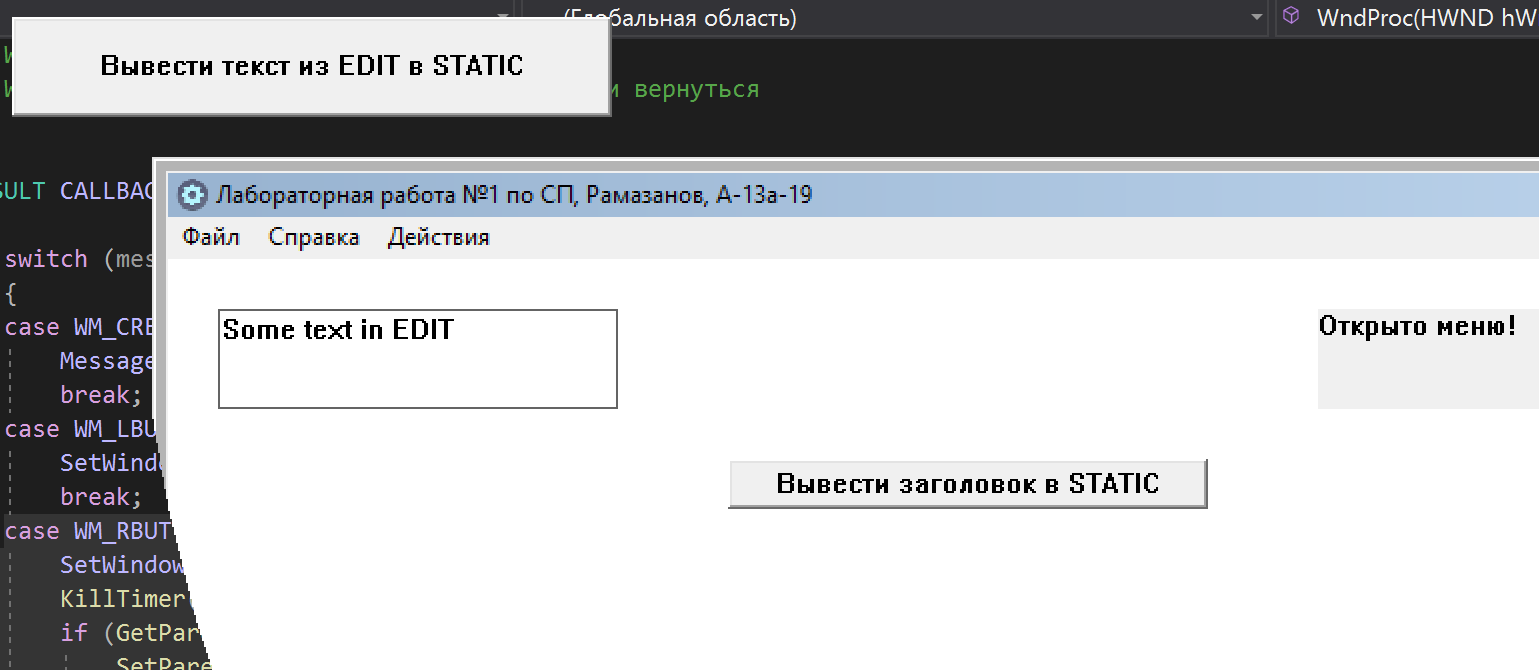
GetWindowText(hWnd, editTextContent, MAX\_LOADSTRING);

SetWindowText(StaticTxt, editTextContent);

}

break;

1. Обработчик пункта меню меняет окно-родителя у первой кнопки, а затем заставляет ее медленно перемещаться по экрану. Нажатие правой кнопки мыши обеспечивает возвращение кнопки в окно программы.

Для реализации этого завел переменные для идентификатора таймера и счетчика:

const static int idTimer = 5; // идентификатор таймера

static int Counter = 0; // счетчик

Написал реакцию на событие WM\_TIMER:

case WM\_TIMER:

{

++Counter;

MoveWindow(Btn1, Counter, Counter, 300, 50, true);

}

break;

А также непосредственно смену родителя, чтобы заставить кнопку двигаться (внутри WM\_COMMAND):

case IDM\_CHANGEPARENT:

{

Counter = 0;

SetTimer(hWnd, idTimer, 30, (TIMERPROC)NULL);

if (GetParent(Btn1) == hWnd) {

SetParent(Btn1, NULL);

}

else {

KillTimer(hWnd, idTimer);

SetParent(Btn1, hWnd);

MoveWindow(Btn1, 250, 150, 300, 50, true);

}

}

break;

И, наконец, добавил в реакцию на нажатие ПКМ смену родителя на главное окно и возврат кнопки в исходное положение:

case WM\_RBUTTONDOWN: // Нажатие на правую кнопку мыши

SetWindowText(StaticTxt, L"Нажатие правой кнопки мыши!");

KillTimer(hWnd, idTimer);

if (GetParent(Btn1) != hWnd) {

SetParent(Btn1, hWnd);

MoveWindow(Btn1, 250, 150, 300, 50, true);

}

break;

1. Оформил окно приложения в форме региона: прямоугольник с эллипсом (нижний край окна) и вырезанным посередине кругом (расположил все управляющие элементы, чтобы они были видны). Проверил работоспособность приложения.

Переменные ширины и высоты окна:

int WWidth = 800;

int WHeight = 600;

Сама функция перерисовки окна (нижняя грань в виде эллипса + вырезан круг посередине):

void WRedrawRgn(HWND hWnd) {

HRGN cutRgn, outRgn;

RECT WRect;

GetWindowRect(hWnd, &WRect);

WWidth = WRect.right - WRect.left; // новая ширина окна

WHeight = WRect.bottom - WRect.top; // новая высота окна

outRgn = CreateEllipticRgn(-10, -WHeight, WWidth + 10, WHeight);

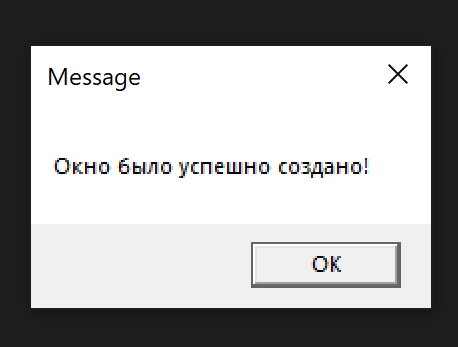
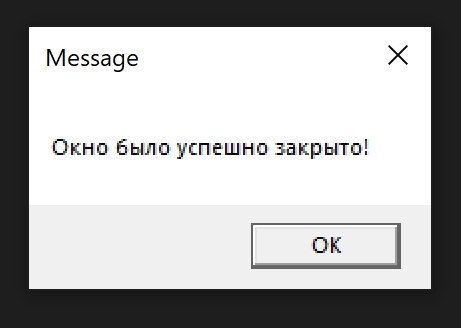
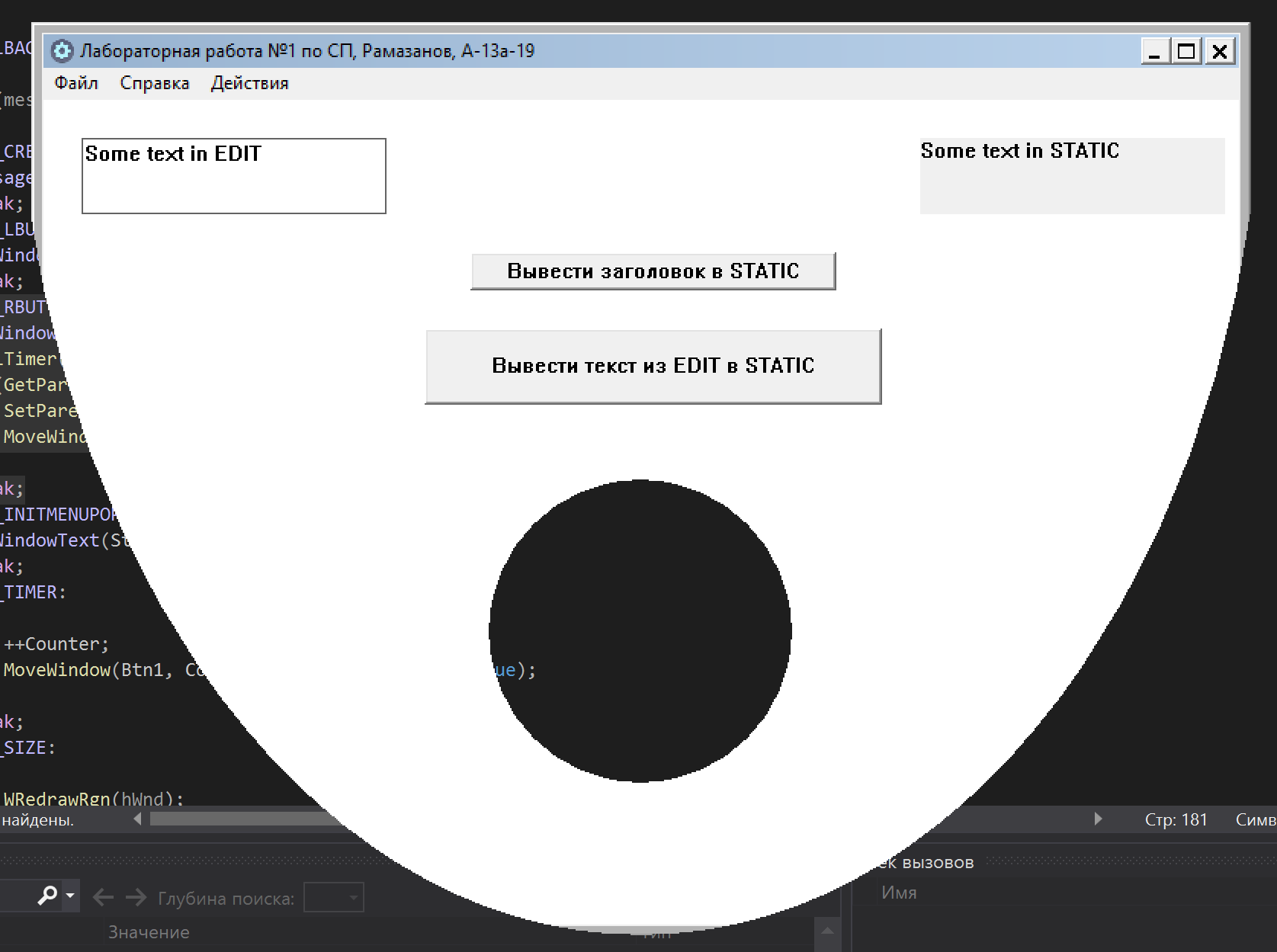
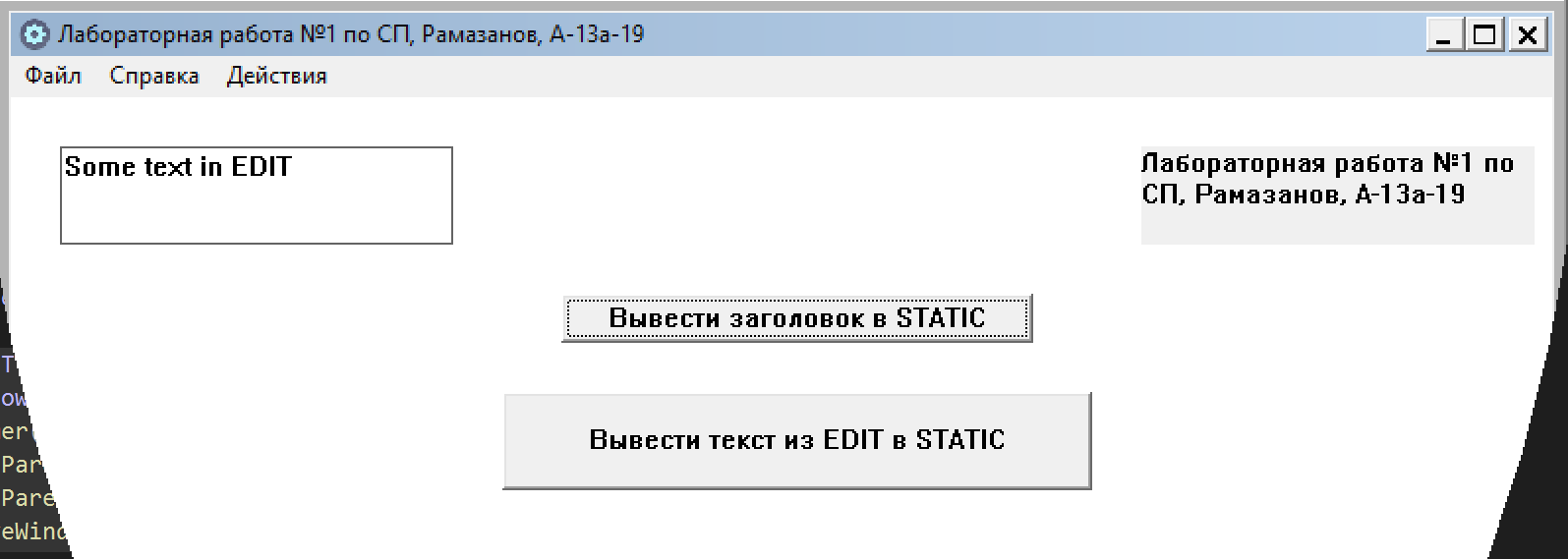
int r = 100; // радиус вырезанного круга

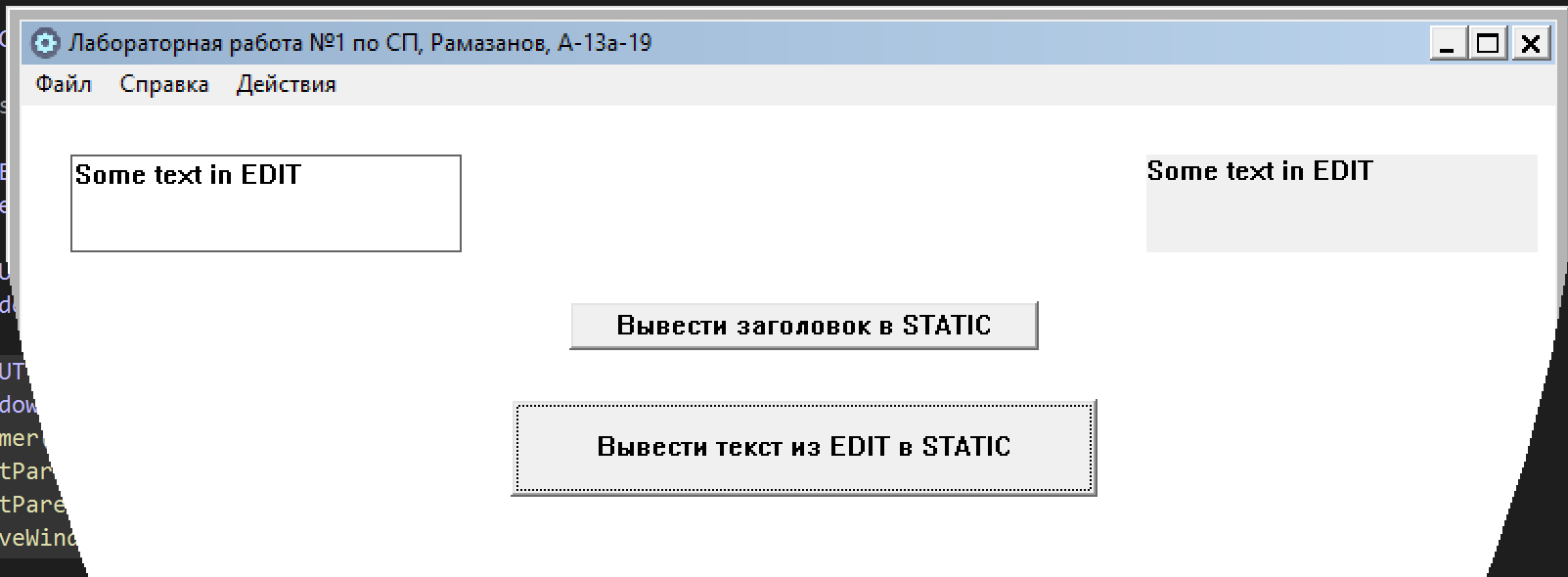
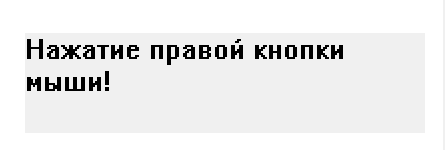
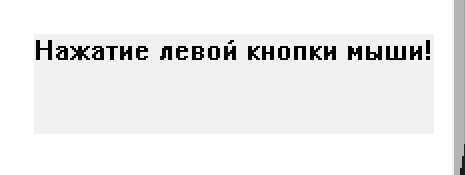
cutRgn = CreateEllipticRgn(WWidth/2-r,WHeight-200-r,WWidth/2+r,WHeight-200+r);

CombineRgn(cutRgn, cutRgn, outRgn, RGN\_XOR);

SetWindowRgn(hWnd, cutRgn, true);

}

Полный код программы доступен по ссылке: <https://github.com/programzan/SystemProgramming/blob/lab1/lab1.cpp>

Здесь же размещены все сопутствующие файлы: <https://github.com/programzan/SystemProgramming/tree/lab1>