**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ**

**Отчет к лабораторной работе №1**

**Дисциплины «Системное программирование»**

**Выполнил студент группы А-13-22**

**Кунавина Мария**

**Преподаватель: Меньшикова К.Г.**

**Москва 2024**

### Подготовка к лабораторной работе

Для выполнения лабораторной работы необходимо ознакомиться (по литературе или в сети) и **сделать краткое описание следующих функций** и **параметров**, а также используемых событий:

1. Получить координаты окна (прямоугольник) GetWindowRect;

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 завершается успешно, возвращаемое значение отличное от нуля.

Если функция не работает, возвращаемое значение - ноль. Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

1. Установить координаты окна SetWindowPos

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.

  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.

);

Значения для hWndInsertAfter:

HWND\_BOTTOM

(HWND) 1

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

(HWND) -2

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

(HWND) 0

Помещает окно вверху Z-порядка.

HWND\_TOPMOST

(HWND) -1

Помещает окно над всеми не самыми верхними окнами. Окно сохраняет свое самое верхнее положение, даже когда оно отключено.

Если функция завершается успешно, возвращаемое значение отличное от нуля.

Если функция не работает, возвращаемое значение - ноль. Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

1. Переместить окно MoveWindow

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 этот параметр имеет значение TRUE, окно получает сообщение. Если параметр имеет значение FALSE, 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 завершается успешно, возвращаемое значение отличное от нуля.

Если функция не работает, возвращаемое значение - ноль. Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

1. Вывод сообщения MessageBox

Например, MessageBox(hWnd,"Текст сообщения"," ",MB\_OK);

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, укажите одно из соответствующих значений.);

Если в окне сообщения есть кнопка «Отмена», функция возвращает значение IDCANCEL, если либо нажата клавиша ESC, либо выбрана кнопка «Отмена». Если в окне сообщения нет кнопки «Отмена», нажатие ESC не даст никакого эффекта - если не присутствует кнопка MB\_OK. Если отображается кнопка MB\_OK и пользователь нажимает ESC, возвращаемое значение будет IDOK.

Если функция не работает, возвращаемое значение - ноль. Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

Если функция завершается успешно, возвращаемое значение - одно из соответствующих значений пункта меню.

1. Создать окно CreateWindow

Например:

HWND hBut1;

hBut1=CreateWindow( TEXT("BUTTON"), TEXT("Вычислить"), WS\_VISIBLE | WS\_CHILD , 100, 100, 80, 26, hWnd, NULL, hInstance, NULL) ;

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, // Указатель на зарегистрированное имя класса

LPCTSTR lpWindowName, // Указатель на имя окна

DWORD dwStyle, // Стиль окна

int x, // Горизонтальная позиция окна

int y, // Вертикальная позиция окна

int nWidth, // Ширина окна

int nHeight, // Высота окна

HWND hWndParent, // Дескриптор родительского или окна владельца

HMENU hMenu, // Дескриптор меню или идентификатор дочернего окна

HANDLE hInstance, // Дескриптор экземпляра приложения

LPVOID lpParam // Указатель на данные создания окна

);

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

Если функция не работает, возвращаемое значение - ПУСТО (NULL). Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

1. Установка идентификатора: SetWindowLong

Пример:

if (hBut1!=NULL) SetWindowLong(hBut1,GWL\_ID,999);

1. 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

);

Если функция завершается успешно, возвращаемое значение - предыдущее значение указанного 32-битного целого числа.

Если функция не работает, возвращаемое значение - ноль. Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

Если предыдущее значение указанного 32-битного целого числа равно нулю и функция завершается успешно, возвращаемое значение равно нулю, но функция не очищает информацию о последней ошибке. Это затрудняет определение успеха или неудачи. Чтобы справиться с этим, вы должны очистить информацию о последней ошибке, вызвав SetLastError с 0 перед вызовом SetWindowLong. Тогда сбой функции будет обозначен возвращаемым значением нуля и результатом GetLastError, отличным от нуля.

1. Функции GetWindowText и SetWindowText

char str[50];

GetWindowText(hEd1,str,50);

SetWindowText(hSt1,str);

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.

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 (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. Чтобы получать расширенные данные об ошибках, вызовите GetLastError.

Эта функция не может извлекать текст из поля элемента редактирования в другой прикладной программе.

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 строчки

);

Если функция завершается успешно, возвращаемое значение отличное от нуля.

Если функция не работает, возвращаемое значение - ноль. Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

1. Функции SetParent и GetParent

SetParent - Изменяет родительское окно указанного дочернего окна.

HWND SetParent

(

HWND hWndChild, // Дескриптор окна, родитель которого изменяется

HWND hWndNewParent // Дескриптор нового родительского окна

);

Если функция завершается успешно, возвращаемое значение - дескриптор предыдущего родительского окна.

Если функция не работает, возвращаемое значение - NULL. Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

GetParent - Извлекает дескриптор родителя или владельца указанного окна.

HWND GetParent(

  HWND hWnd // Дескриптор окна, дескриптор родительского окна которого должен быть получен.

);

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 - ПУСТО (NULL). Чтобы получить расширенную информацию об ошибке, вызовите GetLastError.

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.
3. Функция Sleep (пауза)

Sleep - приостанавливает выполнение текущего потока до истечения времени ожидания.

void Sleep(

  DWORD dwMilliseconds // Интервал времени, на который должно быть приостановлено выполнение, в миллисекундах.

);

1. Функции для работы с регионами:

CombineRgn - Комбинирует два региона между собой

CreateEllipticRgn - Создает регион в виде эллипса или окружности

CreatePolygonRgn - Создает регион в виде многоугольника

CreateRectRgn - Создает прямоугольный регион

CreateRoundRectRgn - Создает регион со скругленными краями из прямоугольной области

SetWindowRgn - Прикрепляет регион к указанному окну

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);

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

int CombineRgn

(

HRGN hrgnDest, // дескриптор результирующей области

HRGN hrgnSrc 1, // дескриптор исходной области

HRGN hrgnSrc 2, // дескриптор исходной области

int fnCombineMode // режим объединения областей

);

Возвращаемое значение указывает тип результирующей области. Это может быть одно из соответстсвующих значений.

Функция 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.

Функция 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.

Функция 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.

Функция 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.

1. Описание событий мыши с параметрами, а также событий WM\_CREATE, WM\_COMMAND, WM\_CLOSE, WM\_DESTROY и WM\_QUIT.

WM\_CREATE посылается тогда, когда программа запрашивает, вызовом какой функции CreateWindowEx или CreateWindow должно быть создано окно. В общем смысле данное событие имеет место быть тогда, когда что-либо в окне нашего приложения создается, указывается или изменяется. Оно говорит о том, что сам объект был создан, он существует, под него выделена память.

WM\_CREATE

lpcs = (LPCREATESTRUCT) lParam; // структура с данными создания

lParam - Указывает на структуру CREATESTRUCT, которая содержит информацию относительно создаваемого окна.

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.

Сообщение WM\_DESTROY посылается тогда, когда окно разрушается. Оно посылается оконной процедуре разрушаемого окна после того, как окно удалено с экрана. Это сообщение посылается сначала разрушаемому окну, а затем дочерним окнам (если таковые имеются) когда они разрушаются. В общем смысле данное событие является предупредительным для системы и подразумевает безопасное окончание работы окна без потери информации и освобождение памяти из-под объекта.

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

WM\_QUIT

nExitCode = (int) wParam;   // код завершения

Параметры nExitCode

Значение wParam. Определяет код завершения, данный в функции PostQuitMessage.

Задания

1. Создать приложение Win32 Project (в Microsoft Visual Studio).

2. Работа с ресурсами приложения:

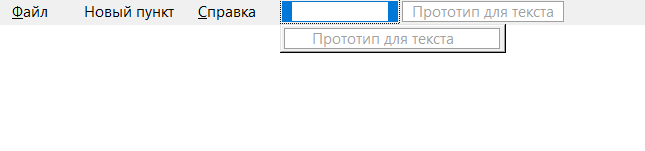
- Изменить заголовок окна (в заголовке указать номер лабораторной работы и фамилию студента);

HWND hWnd = CreateWindowW(szWindowClass, TEXT("Кунавина\_1"), WS\_OVERLAPPEDWINDOW,

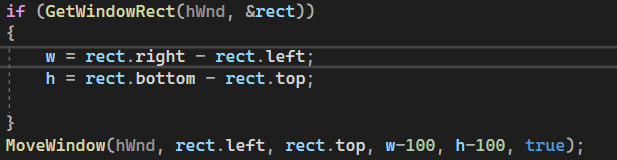
CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

- Добавить в меню новый пункт.

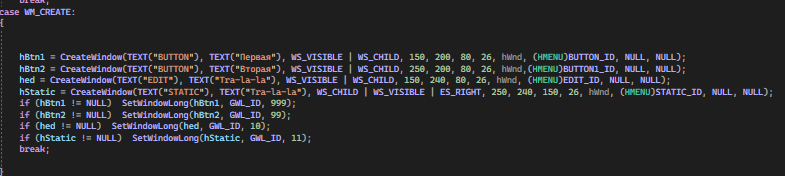
В Файле ресурсов открываем Project.rc-> папку Menu-> IDC\_PROJECT1



3. Задать новые координаты (левый верхний угол, например, (100, 100)) и размеры окна запускаемого приложения (например, 300 и 200). Изменить размеры созданного окна (при этом продумать несколько возможных вариантов, в каком участке кода можно это сделать).

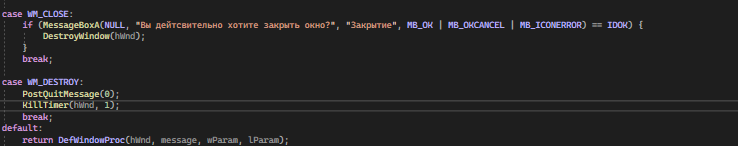


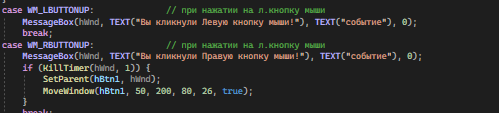
4. Создать управляющие элементы пользовательского интерфейса: две кнопки (BUTTON), поле ввода (EDIT) и поле вывода (STATIC). Для кнопок задать идентификатор управляющего элемента.



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

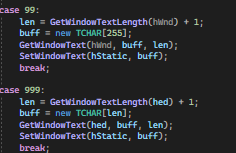
- События: WM\_CREATE, WM\_CLOSE, WM\_DESTROY, нажатие левой кнопки мыши, нажатие правой кнопки мыши, щелчок на добавленном пункте меню должны выводить сообщения о том, какое именно событие произошло;



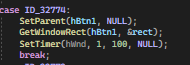


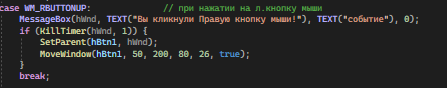
- Первая кнопка BUTTON: текст введенный в поле EDIT должен появиться в поле STATIC;

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



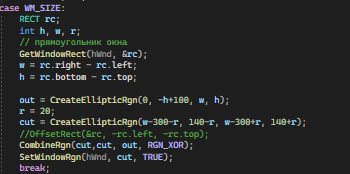
6. Обработчик пункта меню должен: сменить окно - родителя у первой кнопки, а затем заставить ее медленно перемещаться по экрану. Нажатие правой кнопки мыши должно обеспечить возвращение кнопки в окно программы.





ID\_32774- это идентификатор пункта меню, при нажатии на правую кнопку мыши мы останавливаем таймер, и кнопка возвращается на исходные координаты. В WM\_Timer прописаны действия выполняемые каждую секунду, пока он не остановится (кнопка смещается вниз на 10)

7. 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.



8. Изменить название оконной процедуры и проверить работоспособность приложения.







Изменили название на Procedura, вместо  WndProc, ничего не изменилось.

***Листинг:***

#include "framework.h"

#include "Project1.h"

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK Procedura(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_PROJECT1, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_PROJECT1));

MSG msg;

// Цикл основного сообщения:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = Procedura;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_PROJECT1));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_PROJECT1);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, TEXT("Кунавина\_1"), WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

//HWND hBut1, hBut2, hBut3, hBut4;

/\*hBut1 = CreateWindow(TEXT("BUTTON"), TEXT("Первая"), WS\_VISIBLE | WS\_CHILD, 50, 200, 80, 26, hWnd, NULL, hInstance, NULL);

hBut2 = CreateWindow(TEXT("BUTTON"), TEXT("Вторая"), WS\_VISIBLE | WS\_CHILD, 150, 200, 80, 26, hWnd, NULL, hInstance, NULL);

hBut3 = CreateWindow(TEXT("EDIT"), TEXT("Tra-la-la"), WS\_VISIBLE | WS\_CHILD, 50, 240, 80, 26, hWnd, NULL, hInstance, NULL);

hBut4 = CreateWindow(TEXT("STATIC"), TEXT("Tra-la-la"), WS\_VISIBLE | WS\_CHILD, 150, 240, 80, 26, hWnd, NULL, hInstance, NULL);

if (hBut1 != NULL) SetWindowLong(hBut1, GWL\_ID, 999);

if (hBut2 != NULL) SetWindowLong(hBut2, GWL\_ID, 99);

if (hBut3 != NULL) SetWindowLong(hBut3, GWL\_ID, 10);

if (hBut4 != NULL) SetWindowLong(hBut4, GWL\_ID, 11);\*/

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

#define EDIT\_ID 1001

#define BUTTON\_ID 1002

#define BUTTON1\_ID 1003

#define STATIC\_ID 1004

LRESULT CALLBACK Procedura(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static HWND hed;

static HWND hBtn1;

static HWND hBtn2;

static HWND hStatic;

int len;

TCHAR\* buff;

TCHAR\* g;

RECT rect;

int h, w;

switch (message)

{

// При создании окна загружаем указатели мыши

HWND hweIn;

HRGN cut,out;

case WM\_SIZE:

RECT rc;

int h, w, r;

// прямоугольник окна

GetWindowRect(hWnd, &rc);

w = rc.right - rc.left;

h = rc.bottom - rc.top;

out = CreateEllipticRgn(0, -h+100, w, h);

r = 20;

cut = CreateEllipticRgn(w-300-r, 140-r, w-300+r, 140+r);

//OffsetRect(&rc, -rc.left, -rc.top);

CombineRgn(cut,cut, out, RGN\_XOR);

SetWindowRgn(hWnd, cut, TRUE);

break;

case WM\_CREATE:

{

hBtn1 = CreateWindow(TEXT("BUTTON"), TEXT("Первая"), WS\_VISIBLE | WS\_CHILD, 150, 200, 80, 26, hWnd, (HMENU)BUTTON\_ID, NULL, NULL);

hBtn2 = CreateWindow(TEXT("BUTTON"), TEXT("Вторая"), WS\_VISIBLE | WS\_CHILD, 250, 200, 80, 26, hWnd,(HMENU)BUTTON1\_ID, NULL, NULL);

hed = CreateWindow(TEXT("EDIT"), TEXT("Tra-la-la"), WS\_VISIBLE | WS\_CHILD, 150, 240, 80, 26, hWnd, (HMENU)EDIT\_ID, NULL, NULL);

hStatic = CreateWindow(TEXT("STATIC"), TEXT("Tra-la-la"), WS\_CHILD | WS\_VISIBLE | ES\_RIGHT, 250, 240, 150, 26, hWnd, (HMENU)STATIC\_ID, NULL, NULL);

if (hBtn1 != NULL) SetWindowLong(hBtn1, GWL\_ID, 999);

if (hBtn2 != NULL) SetWindowLong(hBtn2, GWL\_ID, 99);

if (hed != NULL) SetWindowLong(hed, GWL\_ID, 10);

if (hStatic != NULL) SetWindowLong(hStatic, GWL\_ID, 11);

break;

}

case WM\_LBUTTONUP: // при нажатии на л.кнопку мыши

MessageBox(hWnd, TEXT("Вы кликнули Левую кнопку мыши!"), TEXT("событие"), 0);

break;

case WM\_RBUTTONUP: // при нажатии на л.кнопку мыши

MessageBox(hWnd, TEXT("Вы кликнули Правую кнопку мыши!"), TEXT("событие"), 0);

if (KillTimer(hWnd, 1)) {

SetParent(hBtn1, hWnd);

MoveWindow(hBtn1, 50, 200, 80, 26, true);

}

break;

/\*case WM\_TIMER:

GetWindowRect(hBtn1, &rect);

w = rect.right - rect.left;

h = rect.bottom - rect.top;

while (not KillTimer(hBtn1, 1)) {

GetWindowRect(hBtn1, &rect);

MoveWindow(hBtn1, rect.left, rect.top + 10, w, h, true);

}

break;\*/

case WM\_COMMAND:

{

int h = 0;

int w = 0;

int count;

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case 99:

len = GetWindowTextLength(hWnd) + 1;

buff = new TCHAR[255];

GetWindowText(hWnd, buff, len);

SetWindowText(hStatic, buff);

break;

case 999:

len = GetWindowTextLength(hed) + 1;

buff = new TCHAR[len];

GetWindowText(hed, buff, len);

SetWindowText(hStatic, buff);

break;

case ID\_32774:

SetParent(hBtn1, NULL);

GetWindowRect(hBtn1, &rect);

SetTimer(hWnd, 1, 100, NULL);

break;

case ID\_32772:

if (GetWindowRect(hWnd, &rect))

{

w = rect.right - rect.left;

h = rect.bottom - rect.top;

}

MoveWindow(hWnd, rect.left, rect.top, w-100, h-100, true);

break;

case ID\_32773:

if (GetWindowRect(hWnd, &rect))

{

w = rect.right - rect.left;

h = rect.bottom - rect.top;

}

MoveWindow(hWnd, rect.left+100, rect.top-50, w, h, true);

break;

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_TIMER:

GetWindowRect(hBtn1, &rect);

MoveWindow(hBtn1, rect.left, rect.top + 10, rect.right - rect.left, rect.bottom - rect.top, true);

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Добавьте сюда любой код прорисовки, использующий HDC...

EndPaint(hWnd, &ps);

}

break;

case WM\_CLOSE:

if (MessageBoxA(NULL, "Вы дейтсвительно хотите закрыть окно?", "Закрытие", MB\_OK | MB\_OKCANCEL | MB\_ICONERROR) == IDOK) {

DestroyWindow(hWnd);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

KillTimer(hWnd, 1);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}