Тема 4. Разработка простейших Win32 API программ

Для выполнения всех практических заданий данной темы необходимо использовать какой-либо компилятор, “умеющий” создавать 32−х разрядный код для систем Windows. Можно использовать компиляторы, встроенные в системы быстрой разработки Windows-приложений Borland Delphi, C++ Builder, MS Visual C++ и др.

Важно понимать, что объектно-ориентированные средства быстрого создания приложений, составляющие суть этих пакетов, при разработке “чистых” WinAPI программ **использовать не надо.** Все дальнейшие примеры в пособии ориентированы на использование Pascal-компилятора пакета Delphi, а примеры использования языка С можно найти в достаточно многочисленной литературе.

Порядок выполнения заданий в системе Delphi:

- запустить систему Delphi и (при необходимости) создать новый проект (меню File/New Application);
- удалить из проекта модуль Unit1 и связанную с ним форму Form1 (меню Project / Remove from Project, выбрать Unit1 и удалить его);
- показать стандартный код Delphi-проекта (меню Project / View Source);
- отредактировать текст проекта в соответствии с приведенной выше базовой структурой API-приложения;
- сохранить текст в файле с именем Project1.dpr, создав для этого новую отдельную папку;
- провести компиляцию проекта и выполнить его (кнопка RUN на панели инструментов);
- закрыть приложение, вернуться в среду разработки и последовательно внести в текст приложения все изменения в соответствии с приводимыми ниже заданиями.

Набор заданий 1. Основы создания Windows-приложений

Задание 1.1. Реализовать простейшую **Windows/API**-программу, которая создает **пустое** главное окно со стандартным поведением. Выполнить программу несколько раз для различных типов курсоров, пиктограмм, заголовков, фонов заполнения окна.

Задание 1.2. Добавить в простейшую программу вывод двух **дочерних** и двух **всплывающих** окон. Исследовать различия в поведении этих окон при их перемещении по экрану и при сворачивании окон.

Задание 1.3. Добавить в простейшую программу вывод в главном окне одного **дочернего** окна, в котором, в свою очередь, выводятся два **своих дочерних** окна. Исследовать различия в поведении этих окон при перемещении и сворачивании.

Задание 1.4. Добавить в простейшую программу возможность отработки оконного сообщения **wm_Create,** которое генерируется при создании оконной структуры вызовом **CreateWindow,** но ДО прорисовки окна на экране. Обработчик этого события в оконной функции должен лишь вывести вспомогательное информационное окно с кратким текстом (например: “Окно создано, но не показано”). Для вывода информационных окон можно использовать API-функцию **MessageBox** (0, текст сообщения, текст в заголовке окна, константа типа). Здесь константа типа определяет набор стандартных кнопок в информационном окне. В самом простом случае достаточно использовать константу **mb_OK,** которая в информационном окне выводит только одну кнопку OK. Поскольку сообщение **wm_Create** возникает при каждом вызове функции **CreateWindow,** в том числе и при создании подчиненных окон, можно оставить в приложении только одно главное окно.

Задание 1.5. Добавить в простейшую программу следующую возможность: при закрытии главного окна должно выводиться информационное

сообщение с подтверждением пользователем факта завершения приложения. Для этого необходимо:

- добавить в оконную функцию обработку сообщения **wm_Close,** которое генерируется системой при закрытии главного окна;
- вывести информационное сообщение с помощью вызова **MessageBox** с использованием константы **mb_OKCancel,** которая позволяет создать в окне 2 кнопки **OK** и **Cancel;**
- для проверки того, какую из двух кнопок в информационном окне выбрал пользователь, можно проверить результат, возвращаемый вызовом **MessageBox**:
  - если выбрана кнопка **OK**, то результат задается константой **idOK;**
  - если выбрана кнопка **Cancel**, то результат задается константой **idCancel.** Проверка выполняется условной инструкцией **if – then.** Если пользователь подтверждает закрытие окна, то вызывается API-функция **DestroyWindow**(дескриптор_окна) для освобождения информационной оконной структуры и завершения цикла обработки сообщений.

Задание 1.6. Изучить по встроенной системе помощи возможности вызова **MessageBox** с другими наборами кнопок и возвращаемых значений. Самый быстрый вызов встроенной помощи: в тексте программы разместить курсор ввода в нужном слове (имя API-функции, системная константа, сообщение, системный тип данных) и нажать клавишу **F1.**

Задание 1.7. Ввести в оконную функцию обработку нажатий левой и правой кнопок мыши (сообщения **wm_LButtonDown** и **wm_RButtonDown**) с выводом в информационном окне сообщения о нажатой кнопке.

Набор заданий 2. Создание и использование меню

Для получения навыков работы с меню последовательно выполнить следующие задания.

Задание 2.1. Программное создание меню. Внести в исходный текст программы следующие изменения:

1. Убрать имя меню в поле описания оконного класса.
2. Сразу после создания главного окна приложения добавить код для программного создания двухуровневого меню (2-3 команды верхнего уровня и 1-2 выпадающих подменю), НЕ забыв объявить необходимое количество переменных-дескрипторов для всех подменю и основного меню.
3. После кода отображения главного окна на экране добавить код для показа меню.
4. Проверить работу созданного приложения.

Задание 2.2. Динамическое подключение к приложению различных меню (например – разноязыковых). В текст предыдущей программы внести следующие изменения:

1. Добавить код для программного создания второго меню со своими дескрипторами, идентификаторами и текстами элементов.
2. При создании главного окна подключить к нему и отобразить только ОДНО (любое) из созданных меню, убрав для второго меню вызовы функций **SetMenu** и **DrawMenuBar.**
3. Для динамического изменения меню добавить в меню две команды и создать для них обработчики в оконной функции, которые должны выполнять два действия:
   - подключение к главному окну необходимого меню по **SetMenu;**
   - отображение полосы меню по **DrawMenuBar.**
4. Проверить работу созданного приложения

Задание 2.3. Ввести в программу возможность динамического изменения меню: добавление новых элементов, удаление элементов, изменение текста элементов, изменение активности элементов. Для управления

элементами меню создать два подменю: одно будет использоваться для отслеживания производимых изменений, второе – для реализации самих изменений.

Набор заданий 3. Вывод графической информации

Задание 3.1. Простейший вывод графических примитивов. Для этого необходимо:

- создать обработчик события wm_Paint с запросом контекста устройства по BeginPaint;
- вывести с помощью стандартных инструментов несколько примитивов:
    - отрезок от текущей точки до заданной: LineTo(контекст, x_конечное, y_конечное); для изменения текущей точки можно использовать вызов MoveToEx(контекст, x, y, nil)
    - эллипс Ellipse(контекст, x_left, y_top, x_right, y_bottom)
    - прямоугольник Rectangle(контекст, x_left, y_top, x_right, y_bottom);
- изменить установленное по умолчанию перо на белое с помощью вызова SelectObject (контекст, GetStockObject (WHITE_PEN)), а кисть – на черную вызовом SelectObject (контекст, GetStockObject (BLACK_BRUSH)) и повторить вывод нескольких разных примитивов;
- установить прозрачное заполнение внутренних частей примитивов, установив в контексте пустую (нулевую) кисть:

  SelectObject (DC, GetStockObject(NULL_BRUSH));

вывести несколько примитивов с прозрачным фоном.

Задание 3.2. Вывод примитивов с помощью цветных перьев. Основные шаги:

- объявление необходимого числа переменных-дескрипторов перьев (тип HPen), причем одна из переменных необходима для сохранения стандартного пера;

- создание необходимого количества перьев с помощью вызова CreatePen, где тип пера задается константами ps_Solid, ps_Dot, ps_Dash;
- выбор одного из перьев в контекст по вызову SelectObject; при выполнении этой операции **первый** раз необходимо сохранить стандартное перо в отдельной переменной;
- вывод примитивов с помощью установленного пера;
- изменение пера с помощью вызова SelectObject и вывод необходимых примитивов;
- после вывода всех примитивов восстановить старое перо вызовом SelectObject;
- при завершении приложения удалить все созданные ранее инструменты.

Для освоения данного механизма создать приложение, которое выводит последовательность расположенных рядом отрезков с помощью предварительно созданного набора перьев. Для этого можно объявить массив из 256 перьев (удобнее использовать индексацию от 0 до 255) и в обработчике события WM_Create в цикле заполнить этот массив цветными перьями. Цвета перьев можно задавать либо случайно, либо по определенной закономерности. Например, циклическое задание цвета вида RGB(i, i, i) при изменении i от 0 до 255 создаст последовательность перьев с постепенным переходом от черного цвета к белому с промежуточными оттенками серого цвета. Аналогично, задание цвета по алгоритму RGB(255 – i, i, i) формирует цветовую гамму с постепенным переходом от красного к голубому.

В обработчике сообщения WM_Paint необходимо:

- запросить контекст с помощью вызова BeginPaint;
- активизировать первое по порядку перо из созданного массива перьев с сохранением стандартного пера;
- вывести первый вертикальный отрезок от начальной точки (0; 0) до конечной точки с помощью вызова LineTo;

- организовать цикл для вывода остальных вертикальных отрезков, внутри которого:
    - активизировать очередное перо из массива перьев;
    - переместить указатель текущей позиции в новую стартовую точку вызовом MoveToEx(контекст, i, 0, nil);
    - вывести новый вертикальный отрезок;
- после завершения цикла восстановить стандартное перо и освободить контекст.

Добавить в обработчик сообщения WM_Destroy цикл для уничтожения массива перьев с помощью вызова DeleteObject(перо).

Задание 3.3. Динамическое изменение цвета изображения. Создать приложение, которое выводит некоторое изображение (например – 5 олимпийских колец) и динамически изменяет его цвет через заданный интервал времени (например – через 1 секунду) с помощью объекта-таймера. Цвет должен выбираться случайно или по какому-либо алгоритму на основе заранее созданного массива из 256 перьев. Для решения задачи необходимо:

- добавить в приложение обработчик события WM_KeyDown с обработкой нажатия какой-либо функциональной клавиши для запуска таймера по вызову SetTimer; здесь же надо установить начальный номер пера;
- добавить в обработчик события WM_KeyDown обработку нажатия другой функциональной клавиши для отключения таймера по вызову KillTimer;
- добавить в оконную функцию обработку события от таймера WM_Timer для программной генерации сообщения WM_Paint с помощью вызова InvalidateRect и изменения номера текущего пера с анализом его значения (при достижении максимального значения надо вернуть номер в начальное значение);

- в обработчике сообщения WM_Paint выполнить обычные действия: запросить контекст, установить прозрачный фон заполнения окружностей по вызову SelectObject(контекст, (NULL_BRUSH)), выбрать очередное перо с сохранением стандартного пера, вывести 5 окружностей, вернуть старой перо и освободить контекст;
- выполнить программу для нескольких значений временного интервала в пределах от 0.1 до 2 секунд.

Задание 3.4. Динамическое изменение структуры изображения. Структура графического изображения, созданного с помощью основных примитивов, определяется перечнем использованных в нем примитивов. В простейшем случае для каждого типа примитива можно ввести свой массив с параметрами данных примитивов. Эта информация используется для перерисовки изображения при изменении его структуры, т.е. набора использованных примитивов.

Необходимо создать приложение для динамического добавления/удаления эллипсов по щелчку левой/правой кнопки мыши. Эллипсы должны прорисовываться в точке, соответствующей щелчку мыши. Размеры эллипсов можно взять фиксированными или устанавливать случайно. Для реализации программы необходимо:

- ввести массив координат эллипсов и переменную-счетчик текущего числа эллипсов;
- в обработчике события wm_Create установить счетчик в ноль;
- в обработчике события wm_LButtonDown извлечь координаты положения мыши, запомнить их в массиве координат, увеличить счетчик и сгенерировать сообщение wm_Paint для перерисовки клиентской части окна;
- в обработчике события wm_Paint выполнить прорисовку эллипсов, используя массив координат и счетчик числа эллипсов;

- в обработчике события wm_RButtonDown уменьшить счетчик на 1 (сделать проверку!) и сгенерировать сообщение wm_Paint для прорисовки клиентской области окна без последнего эллипса;