

Федеральное агентство по образованию
Государственное образовательное учреждение высшего
профессионального образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**С. И. Борисов
М. А. Песков**

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Учебно-методическое пособие

Томск – 2006

Борисов С. И., Песков М. А.

Системное программное обеспечение : учеб. метод. пособие / С. И. Борисов, М. А. Песков. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2006. – 53 с.

Данное учебно-методическое пособие предназначено для студентов специальности 220201 (Управление и информатика в технических системах) и содержит 7 лабораторных работ, которые позволяют научиться разрабатывать оконные приложения для ОС Windows. В ходе выполнения лабораторных работ студенты знакомятся с основными объектами Windows, учатся создавать окна, разрабатывать графический интерфейс пользователя и создавать динамические библиотеки.

© Борисов С. И., Песков М. А., 2006

© Том. гос. ун-т систем упр. и
радиоэлектроники, 2006

Содержание

Введение	5
1 Лабораторная работа №1. Первое окно.....	6
1.1 Цель работы	6
1.2 Предварительные сведения	6
1.3 Задания	8
2 Лабораторная работа №2. Сообщения мыши и клавиатуры.....	11
2.1 Цель работы	11
2.2 Предварительные сведения	12
2.2.1Работа с клавиатурой.....	12
2.2.2Работа с мышью	15
2.3 Задания	18
3 Лабораторная работа №3. Ресурсы	20
3.1 Цель работы	20
3.2 Предварительные сведения	20
3.2.1Создание ресурса типа «Строка»	22
3.2.2Создание ресурса типа «Диалоговое окно»	23
3.2.3Добавление в проект ресурса типа «Растровое изображение» ..	24
3.3 Задания	26
4 Лабораторная работа №4. Шрифты	28
4.1 Цель работы	28
4.2 Предварительные сведения	28
4.3 Задания	33
4.4 Комментарии	34
5 Лабораторная работа №5. Графический редактор	37
5.1 Цель работы	37
5.2 Предварительные сведения	37
5.2.1Битовые образы	39
5.3 Задания	42

6	Лабораторная работа №7. Динамически загружаемые библиотеки (DLL)	43
6.1	Цель работы	43
6.2	Предварительные сведения	43
6.3	Задания	46
	Приложение А	52

Введение

Курс лабораторных работ предназначен для получения практических навыков событийно-ориентированного программирования в среде Windows версий 2000 и старше. Каждая лабораторная работа рассчитана на 6 часов рабочего времени. В качестве инструментальной системы предполагается использовать интегрированную среду разработки Microsoft Visual C++ 6.0.

1 Лабораторная работа №1. Первое окно

1.1 Цель работы

Знакомство с общей схемой функционирования Windows программ на основе программы, созданной при помощи мастера создания приложений. Создание дочерних окон.

1.2 Предварительные сведения

При программировании для Windows вы фактически занимаетесь одним из видов объектно-ориентированного программирования (Object Oriented Programming, OOP). Это наиболее очевидно для объекта, с которым вы в Windows будете большей частью работать, объекта, который дал Windows ее название, объекта, который, как кажется, вскоре приобретет человеческие свойства, объекта, который, быть может, даже будет вам мерещиться, объекта, который известен как "окно". Окна — это прямоугольные области на экране. Окно получает информацию от клавиатуры или мыши пользователя и выводит графическую информацию на своей поверхности.

Окно приложения обычно содержит заголовок (title bar), меню (menu), рамку (sizing border) и иногда полосы прокрутки (scroll bars).

Пользователь рассматривает окна на экране в качестве объектов и непосредственно взаимодействует с этими объектами, нажимая кнопки и переключатели, передвигая бегунок на полосах прокрутки. Достаточно интересно, что положение программиста аналогично положению пользователя. Окно получает от пользователя информацию в виде оконных "сообщений". Кроме этого окно обменивается сообщениями с другими окнами.

Понимание этих сообщений — это один из барьеров, которые нужно преодолеть, чтобы стать программистом для Windows.

Когда говорится: "Windows посылает программе сообщение," — имеется в виду, что Windows вызывает функцию внутри программы. Параметры этой функции описывают параметры сообщения. Эта функция, находящаяся в вашей программе для Windows, называется оконной процедурой.

Вы, несомненно, уже привыкли к мысли, что программа делает вызовы операционной системы. Таким образом, например, программа открывает файл на жестком диске. К чему вы, наверное, еще не можете привыкнуть, так это к тому, что операционная система вызывает программу. Тем не менее, это суть объектно-ориентированной архитектуры Windows. У каждого окна, создаваемого программой, имеется соответствующая оконная процедура. Эта процедура является функцией, которая может находиться либо в самой программе, либо в динамически подключаемой библиотеке. Windows посылает сообщение окну путем вызова оконной процедуры, на основе этого сообщения окно совершает какие-то действия и затем возвращает управление Windows. Более точно, окно всегда создается на основе "класса окна". Класс окна определяет оконную процедуру, обрабатывающую поступающие окну сообщения. Использование класса окна позволяет создавать множество окон на основе одного и того же класса окна и, следовательно, использовать одну и ту же оконную процедуру.

Например, все кнопки во всех программах для Windows созданы на основе одного и того же класса окна. Этот класс связан с оконной процедурой (расположенной в динамически подключаемой библиотеке Windows), которая управляет процессом передачи сообщений всем кнопкам всех окон. В объектно-ориентированном программировании любой "объект" несет в себе сочетание кода и данных. Окно — это объект. Код — это оконная процедура. Данные — это информация, хранимая

оконной процедурой, и информация, хранимая системой Windows для каждого окна и каждого класса окна, которые имеются в системе.

Оконная процедура обрабатывает сообщения, поступающие окну. Очень часто эти сообщения передают окну информацию о том, что пользователь осуществил ввод с помощью клавиатуры или мыши. Таким образом, например, кнопки "узнают" о том, что они нажаты. Другие сообщения говорят окну о том, что необходимо изменить размер окна или о том, что поверхность окна необходимо перерисовать. Когда программа для Windows начинает выполняться, Windows строит для программы очередь сообщений. В этой очереди хранятся сообщения для любых типов окон, которые могли бы быть созданы программой.

Небольшая часть программы, которая называется циклом обработки сообщений, выбирает эти сообщения из очереди и переправляет их соответствующей оконной процедуре. Другие сообщения отправляются непосредственно оконной процедуре, минуя очередь сообщений.

Следует отметить, что окно может иметь на своей поверхности и другие окна. Такие окна называются *дочерними*, а окно, на котором располагаются дочерние окна, является *родительским* по отношению к ним. Для создания дочернего окна необходимо воспользоваться функцией создания окна `CreateWindow`, указав в качестве одно из параметров дескриптор родительского окна.

В операционной системе уже существует набор оконных классов, представляющих собой реализацию различных оконных элементов: кнопок, редакторов строки, статических надписей, списков и т.д. Все эти элементы создаются как дочерние окна.

1.3 Задания

Для создания простейшего Windows-приложения проделайте следующие операции.

- 1 Запустите Microsoft Visual Studio 6.0.
- 2 Выберите пункт меню *File->New*.
- 3 В появившемся окошке выберите вкладку *Project*. Должно появиться диалоговое окно создания приложения.
- 4 Выберите пункт *Win 32 Application*, введите в поле *Project name* имя проекта и выберите в поле *Location* папку, в которой ваш проект будет размещён (рисунок 2.1).

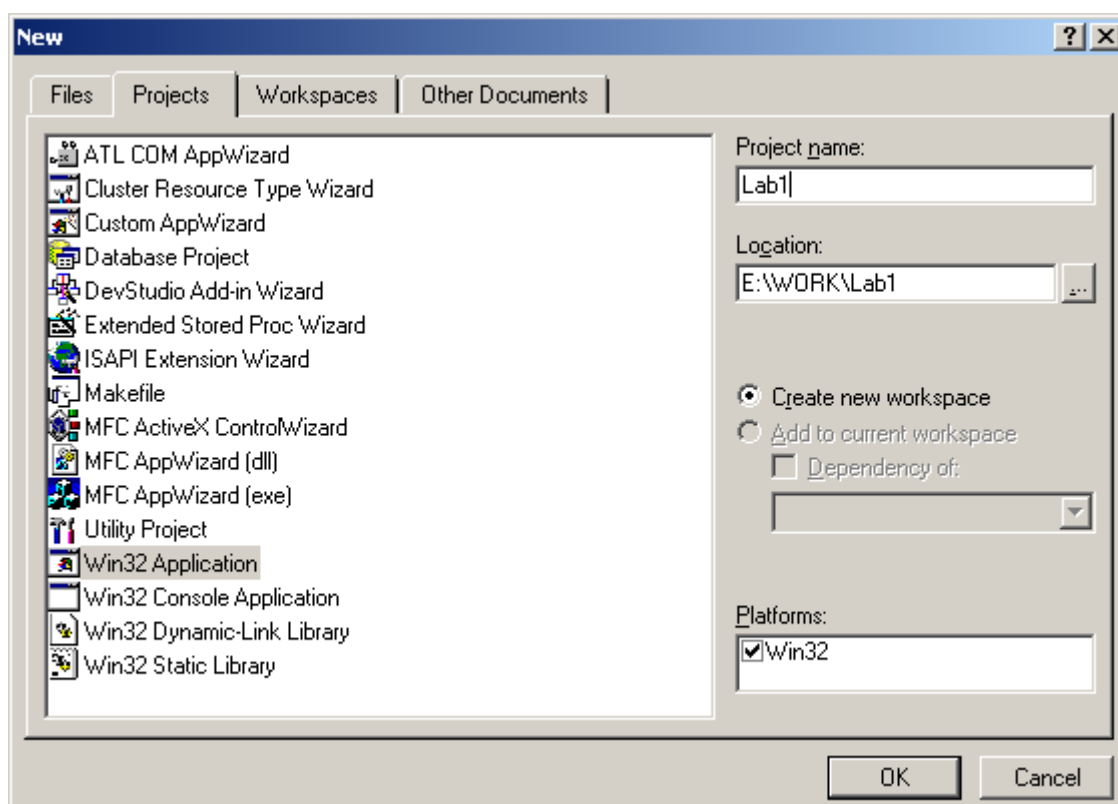


Рис. 1.1 – Диалоговое окно создания приложения

- 5 Нажмите кнопку *Ок*. Должно появиться диалоговое окно выбора типа приложения.
- 6 Выберите пункт *A typical «Hello World» application* (рисунок 1.2) и нажмите кнопку *Finish*.

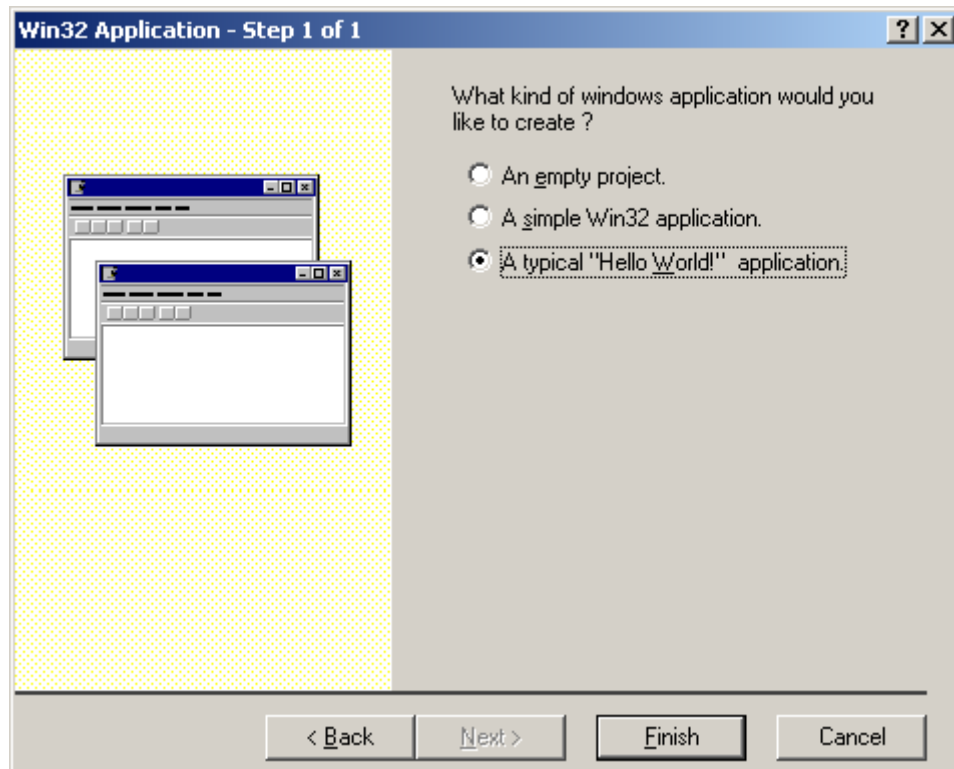


Рис. 1.2 – Диалоговое окно выбора типа приложения

- 7 На вкладке *FileView* откройте главный сpp-файл проекта. Имя главного сpp-файла проекта совпадает с названием проекта (рисунок 3).

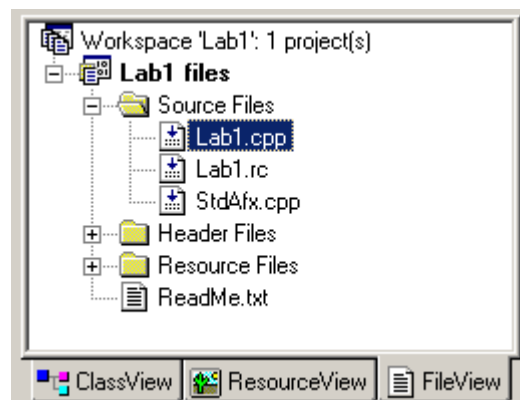


Рисунок 3 – Вкладка *FileView*

- 8 Внимательно изучите полученный код и внесите изменения в него в соответствии с таблицей 2.1.

Таблица 1.1 – Задания на лабораторную работу №1

Номер варианта	Задание
1	Измените стиль главного окна на сочетание стилей WS_BORDER и WS_SYSMENU, добавьте на главное окно дочернее окно типа «Кнопка».
2	Измените стиль главного окна на сочетание стилей WS_OVERLAPPED, WS_SYSMENU и WS_MINIMIZEBOX, добавьте на главное окно дочернее окно типа «Поле редактирования».
3	Измените стиль главного окна на сочетание стилей WS_BORDER и WS_VSCROLL, добавьте на главное окно дочернее окно типа «Список».
4	Измените стиль главного окна на сочетание стилей WS_SYSMENU и WS_DLGFRAME, добавьте на главное окно дочернее окно типа «Комбинированный список».

9 По проделанной работе составьте отчёт. Пример отчёта в приложении А.

2 Лабораторная работа №2. Сообщения мыши и клавиатуры

2.1 Цель работы

Научиться обрабатывать сообщения от клавиатуры и мыши, в том числе сообщения нажатия системных клавиш. Создать простенький редактор строки.

2.2 Предварительные сведения

2.2.1 Работа с клавиатурой

Основанная на сообщениях архитектура Windows идеальна для работы с клавиатурой. Ваша программа узнает о нажатиях клавиш посредством сообщений, которые посылаются оконной процедуре. На самом деле все происходит не столь просто: когда пользователь нажимает и отпускает клавиши, драйвер клавиатуры передает информацию о нажатии клавиш в Windows. Windows сохраняет эту информацию (в виде сообщений) в системной очереди сообщений. Затем она передает сообщения клавиатуры, по одному за раз, в очередь сообщений программы, содержащей окно, имеющее "фокус ввода". Затем программа отправляет сообщения соответствующей оконной процедуре.

Смысл этого двухступенчатого процесса — сохранение сообщений в системной очереди сообщений, и дальнейшая их передача в очередь сообщений приложения — в синхронизации. Если пользователь печатает на клавиатуре быстрее, чем программа может обрабатывать поступающую информацию, Windows сохраняет информацию о дополнительных нажатиях клавиш в системной очереди сообщений, поскольку одно из этих дополнительных нажатий может быть переключением фокуса ввода на другую программу. Информацию о последующих нажатиях следует затем направлять в другую программу. Таким образом Windows корректно синхронизирует такие сообщения клавиатуры.

Клавиатура должна разделяться между всеми приложениями, работающими под Windows. Некоторые приложения могут иметь больше одного окна, и клавиатура должна разделяться между этими окнами в рамках одного и того же приложения. Когда на клавиатуре нажата клавиша, только одна оконная процедура может получить сообщение об этом. Окно, которое получает это сообщение клавиатуры, является окном,

имеющем "фокус ввода". Концепция фокуса ввода тесно связана с концепцией "активного окна". Окно, имеющее фокус ввода — это либо активное окно, либо дочернее окно активного окна. Определить активное окно обычно достаточно просто. Если у активного окна имеется панель заголовка, то Windows выделяет ее. Если у активного окна вместо панели заголовка имеется рамка диалога (это наиболее часто встречается в окнах диалога), то Windows выделяет ее. Если активное окно минимизировано, то Windows выделяет текст заголовка в панели задач. Наиболее часто дочерними окнами являются кнопки, переключатели, флажки, полосы прокрутки и списки, которые обычно присутствуют в окне диалога. Сами по себе дочерние окна никогда не могут быть активными. Если фокус ввода находится в дочернем окне, то активным является родительское окно этого дочернего окна. То, что фокус ввода находится в дочерних окнах, обычно показывается посредством мигающего курсора или каретки.

Если активное окно минимизировано, то окна с фокусом ввода нет. Windows продолжает слать программе сообщения клавиатуры, но эти сообщения выглядят иначе, чем сообщения, направленные активным и еще не минимизированным окнам.

Сообщения, которые приложение получает от Windows о событиях, относящихся к клавиатуре, различаются на "аппаратные" (keystrokes) и "символьные" (characters). Такое положение соответствует двум представлениям о клавиатуре. Во-первых, вы можете считать клавиатуру набором клавиш. В клавиатуре имеется только одна клавиша <A>. Нажатие на эту клавишу является аппаратным событием. Отпускание этой клавиши является аппаратным событием. Но клавиатура также является устройством ввода, генерирующем отображаемые символы.

Клавиша <A>, в зависимости от состояния клавиш <Ctrl>, <Shift> и <CapsLock>, может стать источником нескольких символов. Обычно, этим символом является строчное 'a'. Если нажата клавиша <Shift> или

установлен режим Caps Lock, то этим символом является прописное <A>. Если нажата клавиша <Ctrl>, этим символом является <Ctrl>+<A>. На клавиатуре, поддерживающей иностранные языки, аппаратному событию 'А' может предшествовать либо специальная клавиша, либо <Shift>, либо <Ctrl>, либо <Alt>, либо их различные сочетания. Эти сочетания могут стать источником вывода строчного 'а' или прописного 'А' с символом ударения.

Для сочетаний аппаратных событий, которые генерируют отображаемые символы, Windows посылает программе и оба аппаратных и символьное сообщения. Некоторые клавиши не генерируют символов. Это такие клавиши, как клавиши переключения, функциональные клавиши, клавиши управления курсором и специальные клавиши, такие как <Insert> и <Delete>. Для таких клавиш Windows вырабатывает только аппаратные сообщения.

Когда вы нажимаете клавишу, Windows помещает либо сообщение WM_KEYDOWN, либо сообщение WM_SYSKEYDOWN в очередь сообщений окна, имеющего фокус ввода. Когда вы отпускаете клавишу, Windows помещает либо сообщение WM_KEYUP, либо сообщение WM_SYSKEYUP в очередь сообщений.

Обычно сообщения о "нажатии" и "отпускании" появляются парами. Однако, если вы оставите клавишу нажатой так, чтобы включился автоповтор, то Windows посылает оконной процедуре серию сообщений WM_KEYDOWN (или WM_SYSKEYDOWN) и одно сообщение WM_KEYUP (или WM_SYSKEYUP), когда в конце концов клавиша будет отпущена. Также как и все синхронные сообщения, аппаратные сообщения клавиатуры также становятся в очередь. Вы можете с помощью функции GetMessageTime получить время нажатия и отпускания клавиши относительно старта системы.

Префикс "SYS" в WM_SYSKEYDOWN и WM_SYSKEYUP означает "системное" (system) и относится к аппаратным сообщениям клавиатуры, которые больше важны для Windows, чем для приложений Windows. Сообщения WM_SYSKEYDOWN и WM_SYSKEYUP обычно вырабатываются при нажатии клавиш в сочетании с клавишей <Alt>. Эти сообщения вызывают опции меню программы или системного меню, или используются для системных функций, таких как смена активного окна (<Alt>+<Tab> или <Alt>+<Esc>), или как быстрые клавиши системного меню (<Alt> в сочетании с функциональной клавишей). Программы обычно игнорируют сообщения WM_SYSKEYDOWN и WM_SYSKEYUP и передают их DefWindowProc. Поскольку Windows обрабатывает всю логику Alt-клавиш, то вам фактически не нужно обрабатывать эти сообщения. Ваша оконная процедура в конце концов получит другие сообщения, являющиеся результатом этих аппаратных сообщений клавиатуры (например, выбор меню). Если вы хотите включить в код вашей оконной процедуры инструкции для обработки аппаратных сообщений клавиатуры (как мы это сделаем в программе KEYLOOK, представленной далее в этой главе), то после обработки этих сообщений передайте их в DefWindowProc, чтобы Windows могла по-прежнему их использовать в обычных целях.

2.2.2 Работа с мышью

В Windows для мыши определен набор из 21 сообщения. Однако, 11 из этих сообщений не относятся к рабочей области, и программы для Windows обычно игнорируют их. Если мышь перемещается по рабочей области окна, оконная процедура получает сообщение WM_MOUSEMOVE. Если кнопка мыши нажимается или отпускается внутри рабочей области

окна, оконная процедура получает сообщения, отображённые в таблице 2.1.

Таблица 2.1 – Сообщения от мыши

Кнопка	Нажатие	Отпускание	Двойной щелчок
Левая	WM_LBUTTONDOWN	WM_LBUTTONUP	WM_LBUTTONDBLCLK
Средняя	WM_MBUTTONDOWN	WM_MBUTTONUP	WM_MBUTTONDBLCLK
Правая	WM_RBUTTONDOWN	WM_RBUTTONUP	WM_RBUTTONDBLCLK

Для всех этих сообщений значение параметра `lParam` содержит положение мыши. Младшее слово — это координата *x*, а старшее слово — координата *y* относительно верхнего левого угла рабочей области окна. Вы можете извлечь координаты *x* и *y* из параметра `lParam` с помощью макросов `LOWORD` и `HIWORD`, определенных в заголовочных файлах Windows. Значение параметра `wParam` показывает состояние кнопок мыши и клавиш `<Shift>` и `<Ctrl>`. Вы можете проверить параметр `wParam` с помощью битовых масок, определенных в заголовочных файлах (Таблица 2.2). Префикс МК означает "клавиша мыши" (`mouse key`).

Таблица 3 – Битовые маски, отражающие состояние клавиш

Маска	Значение
МК_LBUTTON	Левая кнопка нажата
МК_MBUTTON	Средняя кнопка нажата
МК_RBUTTON	Правая кнопка нажата
МК_SHIFT	Клавиша «Shift» нажата
МК_CONTROL	Клавиша «Ctrl» нажата

При движении мыши по рабочей области окна, Windows не вырабатывает сообщение `WM_MOUSEMOVE` для всех возможных положений мыши. Количество сообщений `WM_MOUSEMOVE`, которые получает ваша программа, зависит от устройства мыши и от скорости, с которой ваша

оконная процедура может обрабатывать сообщения о движении мыши. Вы получите хорошее представление о темпе получения сообщений WM_MOUSEMOVE, когда поэкспериментируете с представленной ниже программой CONNECT. Если вы щелкните левой кнопкой мыши в рабочей области неактивного окна, Windows сделает активным окно, в котором вы произвели щелчок, и затем передаст оконной процедуре сообщение WM_LBUTTONDOWN. Если ваша оконная процедура получает сообщение WM_LBUTTONDOWN, то ваша программа может уверенно считать, что ее окно активно. Однако, ваша оконная процедура может получить сообщение WM_LBUTTONUP, не получив вначале сообщения WM_LBUTTONDOWN. Это может случиться, если кнопка мыши нажимается в одном окне, мышь перемещается в ваше окно, и кнопка отпускается. Аналогично, оконная процедура может получить сообщение WM_LBUTTONDOWN без соответствующего ему сообщения WM_LBUTTONUP, если кнопка мыши отпускается во время нахождения в другом окне.

Из этих правил есть два исключения:

- 1 Оконная процедура может "захватить мышь" (capture the mouse) и продолжать получать сообщения мыши, даже если она находится вне рабочей области окна. Позднее в этой главе вы узнаете, как захватить мышь.
- 2 Если системное модальное окно сообщений или системное модальное окно диалога находится на экране, никакая другая программа не может получать сообщения мыши. Системные модальные окна сообщений и диалога запрещают переключение на другое окно программы, пока оно активно. (Примером системного модального окна сообщений является окно, которое появляется, когда вы завершаете работу с Windows.)

2.3 Задания

Создать программу, включающую в себя два окна. В первом окне реализовать задание из таблицы 2.3, во втором - редактор строки, работающий следующим образом.

- 1 При нажатии на любую алфавитно-цифровую клавишу, в конец вводимой строки добавляется символ, соответствующий этой клавиши в текущем регистре.

При нажатии на кнопку «Back space» при условии, что строка не пуста, удаляется последний символ в строке.

Таблица 2.3 – Задания на лабораторную работу №2

Номер варианта	Задание
1	Обработать событие WM_MOUSEMOVE. При перемещении курсора мыши над окном, в окне отображается координата X курсора мыши и состояние клавиши <i>Shift</i> . Обработать событие WM_KEYDOWN. При нажатии на любую клавишу на клавиатуре выводить код этой клавиши, а в случае, если нажатая клавиш алфавитно-цифровая, то и сам символ, соответствующий этой клавиши.
2	Обработать событие WM_LBUTTONDOWN. При нажатии левой кнопкой мыши в области окна, в окне отображается координата Y курсора мыши и состояние клавиши <i>Control</i> . Обработать событие WM_KEYUP. При отпуске любой кнопки на клавиатуре выводить код этой клавиши, а в случае, если нажатая клавиш алфавитно-цифровая, то и сам символ, соответствующий этой клавиши.

Таблица 2.3 – Задания на лабораторную работу №2

Номер варианта	Задание
3	Обработать событие WM_LBUTTONDOWNBLCLK. При двойном щелчке левой кнопкой мыши в области окна, в окне отображается координата Y курсора мыши и состояние клавиши <i>Shift</i> . Обработать событие WM_SYSKEYDOWN. При нажатии на любую кнопку на клавиатуре в сочетании с клавишей <i>Alt</i> выводить код этой клавиши, а в случае, если нажатая клавиш алфавитно-цифровая, то и сам символ, соответствующий этой клавиши.
4	Обработать событие WM_MOUSEMOVE. При перемещении курсора мыши над окном, в окне отображается координата X курсора мыши и состояние клавиши <i>Shift</i> . Обработать событие WM_SYSKEYUP. При отпускании любой клавиши на клавиатуре в сочетании с клавишей <i>Alt</i> выводить код этой клавиши, а в случае, если нажатая клавиш алфавитно-цифровая, то и сам символ, соответствующий этой клавиши.

3 Лабораторная работа №3. Ресурсы

3.1 Цель работы

Познакомиться с типами ресурсов Windows, способами их загрузки. Научиться пользоваться редактором ресурсов. Узнать способ отображения битмапа в окне.

3.2 Предварительные сведения

В большинство программ для Windows включаются пользовательские значки, которые Windows выводит на экран в левом верхнем углу строки заголовка окна приложения. Кроме этого Windows выводит на экран значок программы в списках программ меню «Пуск», или в панели задач в нижней части экрана, или в списке программы Windows Explorer. Некоторые программы — наиболее известными из которых являются графические программы для рисования, например Windows Paint, используют собственные курсоры мыши для отражения различных действий программы. В очень многих программах для Windows используются окна меню и диалога. Вместе с полосами прокрутки окна меню и диалога — это основа стандартного пользовательского интерфейса Windows.

Значки, курсоры, окна меню и диалога связаны между собой. Все это виды ресурсов (resources) Windows. Ресурсы являются данными, и они хранятся в .EXE файле программы, но расположены они не в области данных, где обычно хранятся данные исполняемых программ. Таким образом, к ресурсам нет непосредственного доступа через переменные, определенные в исходном тексте программы. Они должны быть явно загружены из файла с расширением .EXE в память.

Когда Windows загружает в память код и данные программы для ее выполнения, она обычно оставляет ресурсы на диске. Только тогда, когда

Windows нужен конкретный ресурс, она загружает его в память. Действительно, вы могли обратить внимание на такую динамическую загрузку ресурсов при работе с Windows-программами. Когда вы первый раз вызываете окно диалога программы, Windows обычно обращается к диску для копирования ресурса окна диалога из файла с расширением .EXE программы в оперативную память.

Существуют следующие виды ресурсов:

- значки (icons)
- курсоры (cursors)
- битовые образы (bitmaps)
- символьные строки (character strings)
- меню (menus)
- быстрые комбинации клавиш (keyboard accelerators)
- окна диалога (dialog boxes)
- ресурсы, определяемые пользователем (user defined resources)

При создании программы ресурсы определяются в файле описания ресурсов (resource script), который представляет собой ASCII-файл с расширением .RC. Файл описания ресурсов может содержать представление ресурсов в ASCII-кодах, а также может ссылаться и на другие файлы (ASCII или бинарные файлы), в которых содержатся остальные ресурсы. С помощью компилятора ресурсов (файл RC.EXE) файл описания ресурсов компилируется и становится бинарным файлом с расширением .RES. Задав в командной строке LINK файл с расширением .RES, вы можете заставить компоновщик включить скомпилированный файл описания ресурсов в файл с расширением .EXE программы вместе с обычными кодом и данными программы из файлов с расширением .OBJ и .LIB.

В описание ресурсов вы можете добавить заголовочный файл с расширением `.h`. Кроме того этот заголовочный файл обычно также включается в файл с исходным текстом программы на C и содержит определения идентификаторов, которые программа использует для ссылки на ресурсы. В списке зависимостей возможно появление и других файлов. Это те файлы, на которые в описании ресурсов имеется ссылка. Обычно это бинарные файлы со значками, курсорами и битовыми образами.

Компилятор ресурсов использует препроцессор, который может учитывать добавленные и удаленные константы, определять символы ограничителей комментариев `/*` и `*/`, и директивы препроцессора C `#define`, `#undef`, `#ifdef`, `#ifndef`, `#include`, `#if`, `#elif`, `#else` и `#endif`.

3.2.1 Создание ресурса типа «Строка»

Для добавления ресурса типа «строка», необходимо проделать нижеследующие операции.

- 1 На вкладке *ResourceView* выберите в дереве подпункт *String Table->String Table*.

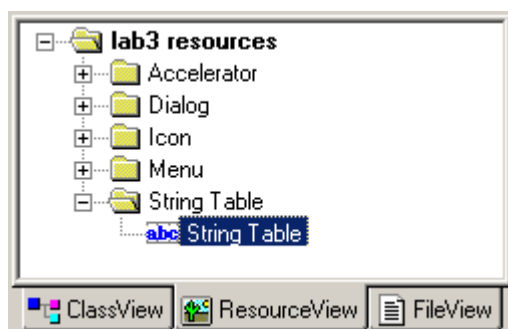


Рис. 3.1 — Таблица строк в файле ресурса

- 2 В появившейся справа таблице строк щёлкните два раза на пустой строке, которая находится в самом низу таблицы (рисунок 3.2).

ID	Value	Caption
IDS_APP_TITLE	103	lab3
IDS_HELLO	106	Hello 'World!
IDC_LAB3	109	LAB3

Рис 3.2 — Таблица строк

3 В появившемся диалоговом окне задайте название идентификатора строки и текст новой строки (рисунок 3.3). В файл ресурса добавится новая строка, которой будет присвоен идентификатор, описанный в файле *resource.h*, значение которого будет сгенерировано автоматически.

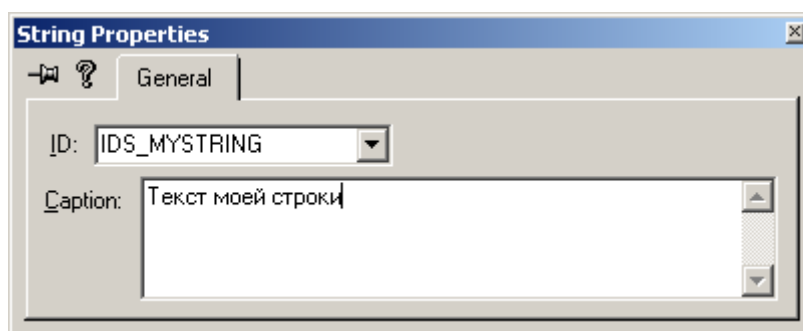


Рис. 3.3 — Ввод новой строки

3.2.2 Создание ресурса типа «Диалоговое окно»

Для добавления ресурса типа «диалоговое окно» необходимо на вкладке *ResourceView* щёлкнуть правой клавишей на пункте *Dialog* и выбрать пункт *Insert Dialog* (рисунок 3.4). Справа появится специальный редактор, при помощи которого можно изменить параметры диалогового окна и добавить необходимые элементы (рисунок 3.5).

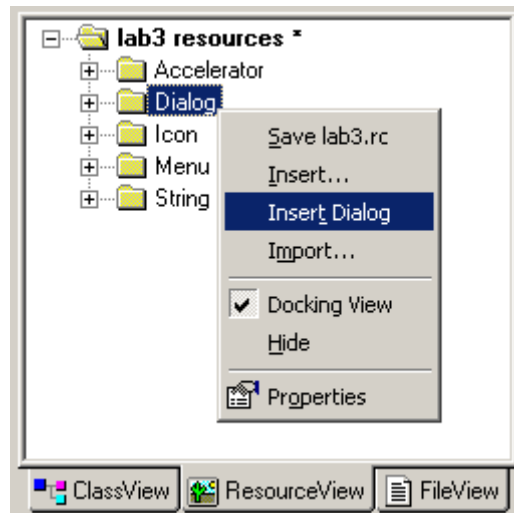


Рис. 3.4 – Добавление нового диалогового окна

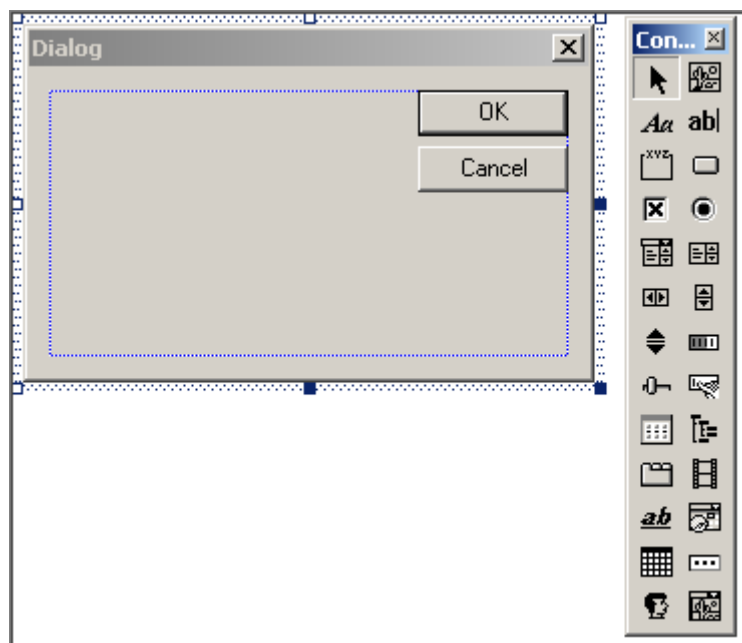


Рис. 3.5 – Редактор диалогового окна

3.2.3 Добавление в проект ресурса типа «Растровое изображение»

В файл ресурса возможно добавить либо уже готовое растровое изображение, указав имя bmp-файла, либо нарисовав свой рисунок. В первом случае, в файл ресурса добавится только ссылка на файл, а во втором – непосредственно двоичное представление растрового изображения. Для того, чтобы добавить в ресурс ссылку на файл с

картинкой, необходимо на вкладке *ResourceView* щёлкнуть правой кнопкой мыши и выбрать подпункт *Import* (Рис. 3.6). Затем – открыть подключаемый файл с картинкой.

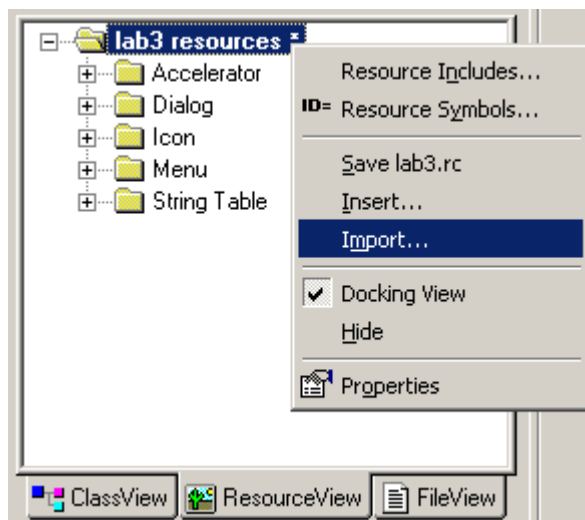


Рис. 3.6 – Подключение файла к ресурсу

Для встраивания файла в ресурс необходимо проделать следующие операции.

- 1 На вкладке *ResourceView* щёлкнуть правой кнопкой мыши и выбрать подпункт *Insert* (рисунок 3.7).

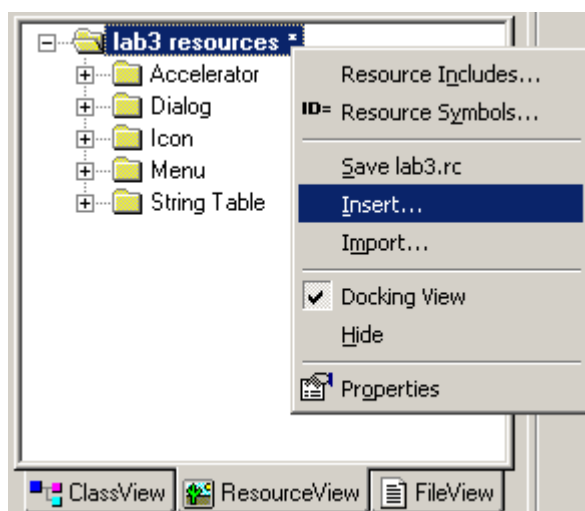


Рис. 3.7 – Добавление в ресурс нового элемента

- 2 В появившемся диалоговом окне выбрать пункт *Bitmap* (рисунок 3.8). Справа появится редактор рисунка (рисунок 3.9).

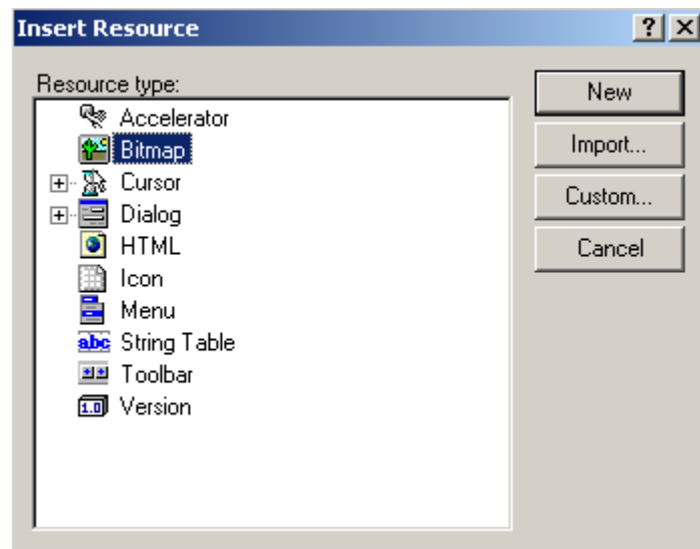


Рис. 3.8 – Добавление в ресурс растрового изображения

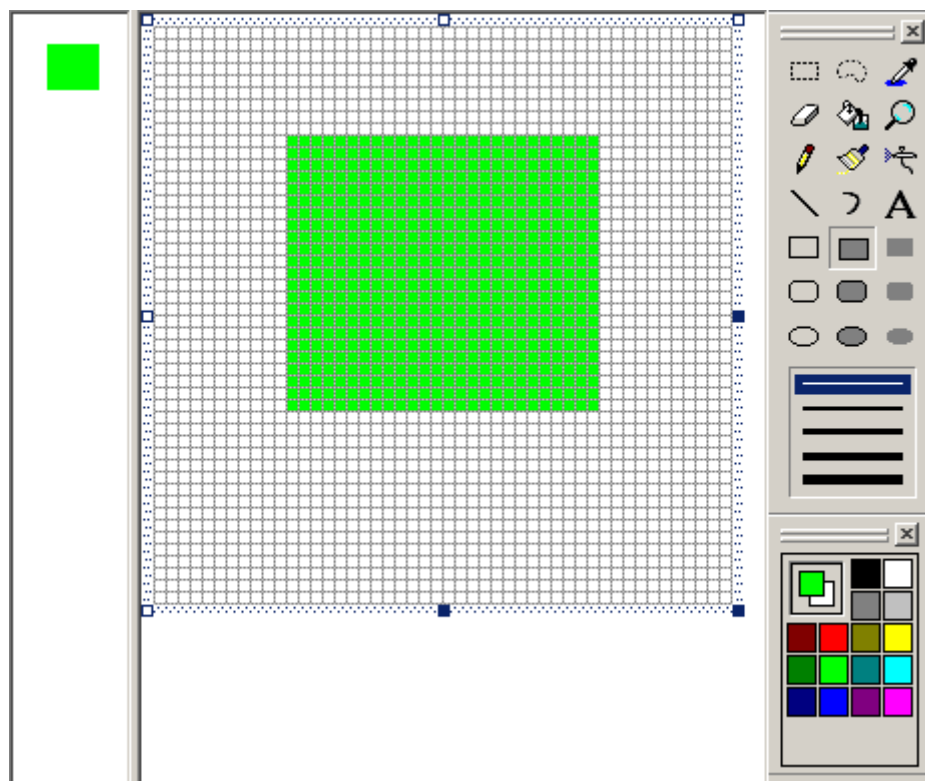


Рис. 3.9 – Редактор растровых изображений

3.3 Задания

1 Добавьте в файл ресурса:

- строку, содержащую вашу фамилию, имя и отчество;
- растровое изображение (битмап);

- ресурсы, перечисленные в таблице 5, в зависимости от варианта;
 - диалоговое окно.
- 2 Прodelать манипуляции над ресурсами, описанные в таблице 3.
 - 3 В главное окно программы добавить кнопку, в обработчике клика которого сделать отображение диалогового окна.

Таблица 3.1 – Задания на лабораторную работу №3

Номер варианта	Ресурс	Манипуляции над ресурсом
1	Иконка	Сделать иконкой главного окна.
2	Курсор	Сделать курсором, который будет отображаться над диалоговым окном.
3	Главное меню	Сделать главным меню главного окна.
4	Всплывающее меню	Отображать при щелчке правой кнопкой мыши.

4 Лабораторная работа №4. Шрифты

4.1 Цель работы

Научиться работать с различными шрифтами, зарегистрированными в системе. Изменять различные параметры отображения текста, получение метрик шрифтов, и использование в прикладных целях различных шрифтов. Перечисление шрифтов.

4.2 Предварительные сведения

Очень важно понимать разницу между окном приложения и его рабочей областью: рабочая область — это часть всего окна приложения, в верхней части которой нет строки заголовка, у которой нет рамки окна, нет строки меню, нет полос прокрутки. Короче говоря, рабочая область — это часть окна, на которой программа может рисовать и представлять визуальную информацию для пользователя. Вы можете делать с рабочей областью вашей программы почти все, что захотите — все, за исключением того, что у вас отсутствует возможность задать ей определенный размер или оставить этот размер неизменным во время работы вашей программы. Ваша программа должна учитывать размер окна и использовать рациональные способы работы с ним.

При работе в MS-DOS программа, использующая полноэкранный режим вывода информации, может выводить текст в любую часть экрана. То, что программа вывела на экран, там и останется и никуда таинственно не исчезнет. Программе уже не нужна информация, необходимая для повторного вывода информации на экран. Если другая программа (например, резидентная) закроет часть экрана, тогда эта резидентная программа и должна, после завершения своей работы, восстановить содержимое экрана. В Windows можно выводить информацию только в рабочую область окна, и вы не можете быть уверены в том, что там что-

нибудь будет оставаться, до тех пор пока ваша программа специально не выведет что-нибудь поверх. Например, окно диалога другого приложения может перекрыть часть вашей рабочей области. Хотя Windows будет пытаться сохранить и восстановить область экрана под окном диалога, это не всегда получается. После того как окно диалога удаляется с экрана, Windows выдаст запрос, требующий, чтобы ваша программа перерисовала эту часть рабочей области.

Windows — это операционная система, управляемая сообщениями. Windows уведомляет приложения о различных событиях путем постановки синхронных сообщений в очередь сообщений приложения или путем отправки асинхронных сообщений соответствующей процедуре окна. Посылая синхронное сообщение `WM_PAINT`, Windows уведомляет оконную процедуру о том, что часть рабочей области окна необходимо обновить.

Для рисования в рабочей области вашего окна, вы используете функции графического интерфейса устройства. В Windows имеется несколько функций GDI для вывода строк текста в рабочей области окна. Например, функция `TextOut`. Формат этой функции следующий:

```
TextOut(hdc, x, y, psString, iLength);
```

Функция `TextOut` выводит на экран строку символов. Параметр `psString` — это указатель на строку символов, а `iLength` — длина строки символов. Параметры `x` и `y` определяют начальную позицию строки символов в рамках рабочей области. Параметр `hdc` — это "дескриптор контекста устройства", являющийся важной частью GDI. Практически каждой функции GDI в качестве первого параметра необходим этот дескриптор.

Дескриптор — это просто число, которое Windows использует для внутренней ссылки на объект. Вы получаете дескриптор от Windows и затем используете этот дескриптор в разных функциях. Дескриптор контекста устройства — это паспорт вашего окна для функций GDI. Этот дескриптор дает вам полную свободу при рисовании в рабочей области вашего окна, и вы можете сделать ее такой, как пожелаете. Контекст устройства фактически является структурой данных, которая внутренне поддерживается GDI. Контекст устройства связан с конкретным устройством вывода информации, таким как принтер, плоттер или дисплей. Что касается дисплея, то в данном случае контекст устройства обычно связан с конкретным окном на экране.

Когда программе необходимо начать рисование, она должна получить дескриптор контекста устройства. После окончания рисования программа должна освободить дескриптор. Когда программа освободит дескриптор, он становится недействительным и не должен далее использоваться. Во время обработки каждого отдельного сообщения программа должна получить и освободить дескриптор. Получить дескриптор контекста экрана можно двумя методами.

Первый метод используется при обработке сообщений `WM_PAINT`. Применяются две функции: `BeginPaint` и `EndPaint`. Для этих двух функций требуется дескриптор окна (передаваемый в оконную процедуру в качестве параметра) и адрес переменной типа структуры `PAINTSTRUCT`. Во время обработки сообщения `WM_PAINT` оконная процедура сначала вызывает `BeginPaint` для заполнения полей структуры. Возвращаемым значением функции `BeginPaint` является дескриптор контекста устройства типа `HDC`. Тип данных `HDC` определяется как 32-разрядное беззнаковое целое. Затем программа может использовать функции GDI,

например `TextOut`. Вызов функции `EndPaint` освобождает дескриптор контекста устройства.

Типовой процесс обработки сообщения `WM_PAINT` выглядит следующим образом:

```
case WM_PAINT:
hdc=BeginPaint(hwnd, &ps);
[использование функций GDI]
EndPaint(hwnd, &ps);
return 0;
```

Таким образом, функции `BeginPaint` и `EndPaint` являются скобками для рисования примитивов GDI.

Вы также можете получить дескриптор контекста устройства при помощи функции `GetDC`, если хотите рисовать в рабочей области при обработке отличных от `WM_PAINT` сообщений, или если вам необходим дескриптор контекста устройства для других целей, например, для получения информации о самом контексте устройства. Вызывайте `GetDC` для получения дескриптора контекста устройства и `ReleaseDC`, если он вам больше не нужен:

```
hdc=GetDC(hwnd);
[использование функций GDI]
ReleaseDC(hwnd, hdc);
```

Контекст устройства также определяет шрифт, который Windows использует при выводе текста в рабочую область. По умолчанию задается так называемый "системный шрифт" или (используя идентификатор заголовочных файлов Windows) `SYSTEM_FONT`. Системный шрифт — это

шрифт, который Windows использует для текста заголовков, меню и окон диалога.

Для вывода на экран нескольких строк текста с использованием функции `TextOut`, вам необходимо задать размеры символов шрифта. Следующие друг за другом строки текста вы размещаете на экране с учетом высоты символа, а колонки текста в рабочей области — исходя из усредненной ширины символов шрифта. Размеры символов можно получить с помощью вызова функции `GetTextMetrics`. Для функции `GetTextMetrics` требуется дескриптор контекста устройства, поскольку ее возвращаемым значением является информация о шрифте, выбранном в данное время в контексте устройства. Windows копирует различные значения метрических параметров текста в структуру типа `TEXTMETRIC`:

```
TEXTMETRIC tm;
hdc = GetDC(hwnd);
GetTextMetrics(hdc, &tm);
ReleaseDC(hwnd, hdc);
```

Теперь вы можете проанализировать значения в структуре текстовых размеров и, возможно, сохранить несколько из них для использования в будущем.

Вы можете получить дескриптор стандартного шрифта, вызывая функцию:

```
HFONT hFont = GetStockObject(iFont);
```

где параметр `iFont` — один из нескольких идентификаторов, только два из которых обычно используются. Вы можете затем выбрать шрифт в контекст устройства:

```
SelectObject(hdc, hFont);
```


Функция `GetStockObject` также используется для получения стандартных перьев и кистей; функцию, а `SelectObject` – для выбора перьев, кистей, битовых образов и регионов в контекст устройства.

Чтобы создать свой нестандартный шрифт, необходимо воспользоваться функцией `CreateFontIndirect`, входным параметром которой является указатель на структуру типа `LOGFONT`.

```
LOGFONT lf;
[Заполнение полей]
HFONT hFont = CreateFontIndirect(&lf);
HFONT hFontOld = (HFONT) SelectObject(hdcMem,
CreateFontIndirect(&lf));
[Вывод текста созданным шрифтом]
SelectObject(hdcMem, hFontOld);
DeleteObject(hFont);
```

При использовании функции `SelectObject` необходимо запомнить старый шрифт и вернуть его после окончания использования нового шрифта. Созданный же шрифт необходимо удалить после использования при помощи функции `DeleteObject`. Это же касается при использовании *любых* объектов GDI.

4.3 Задания

- 1 Добавить на поверхность окна список.
- 2 Перечислить зарегистрированные в системе шрифты в списке (`ListBox`).
- 3 Нарисовать в окне фигуру, указанную в таблице 4.1 из слов «Windows Font» шрифтом, выбранным в списке.
- 4 Вывести в дочернем окне метрики шрифта, выбранного в списке.

- 5 Создать дочернее окно типа «Редактор». При помощи функции `GetStockObject` получить стандартный шрифт, указанный в таблице 6 и задать его редактору.

Таблица 4.1 – Задания на лабораторную работу № 4

Номер варианта	Фигура	Шрифт
1	Парабола	<code>ANSI_FIXED_FONT</code>
2	Синусоида	<code>ANSI_VAR_FONT</code>
3	Спираль	<code>SYSTEM_FONT</code>
4	Гипербола	<code>OEM_FIXED_FONT</code>

4.4 Комментарии

Построение геометрической фигуры можно выполнить по следующему алгоритму.

- 1 Задать первоначальную точку, сделать её текущей точкой.
- 2 Вычислить производную в текущей точке.
- 3 По производной вычислить угол наклона буквы.
- 4 Вывести очередной символ.
- 5 Узнать ширину и высоту выведенного символа.
- 6 Посчитать координаты следующего символа по формуле:

$$\text{координата } x \text{ следующего символа} = \text{ширина текущего символа} * \cos(\text{текущего символа}) + \text{высота текущего символа} * \sin(\text{текущего символа})$$
- 7 Вычислить, построена ли геометрическая фигура (вышли ли текущие координаты за пределы области вывода функции).
- 8 Если геометрическая фигура не построена, то перейти к пункту 2, иначе завершить построения.

Можете воспользоваться нижеследующим примером – функцией, которая выводит на экран слова «Windowd Font» в виде окружности.

```

void DrawCircle(HDC hdc) {
HFONT hfont = GetCurrentObject(hdc, OBJ_FONT);
#define RADIUS 250
#define CENTRX 270
#define CENTRY 270
#define BETWEEN_SYM 3

char str[] = "Windows Font ";
int num_sym = 0;

for (int y_orientation = -1; y_orientation <= 1;
y_orientation+=2)
for (int x = - (RADIUS - 1); x < RADIUS; ) {
// По производной вычисляем угол наклона
float alpha;
alpha = atan(-(x/sqrt(RADIUS*RADIUS - x*x)));
if (y_orientation == -1) alpha -= (3.14);

// Создаём новый шрифт и задаём ему угол наклона
LOGFONT lf;
GetObject(hfont, sizeof(lf), &lf);
lf.lfEscapement = alpha*1800 / 3.14;
HFONT hfont_new = CreateFontIndirect(&lf);
SelectObject(hdc, hfont_new);

// Выводим очередной символ
int y = sqrt(RADIUS*RADIUS - x*x);
TextOut(hdc, x*y_orientation + CENTRX,
CENTRY - y*y_orientation,

```

```
&str[num_sym], 1);

// Вычисляем ширину выведенного символа и координаты
следующего символа
SIZE sz;
GetTextExtentPoint(hdc, &str[num_sym], 1, &sz);
x += abs(sz.cx * cos(alpha) + sin(alpha)) +
BETWEEN_SYM;

// Удаляем созданный шрифт
SelectObject(hdc, hfont);
DeleteObject(hfont_new);

// Выбираем очередной символ
num_sym++;
if (num_sym == strlen(str)) num_sym = 0;
}
}
```

5 Лабораторная работа №5. Графический редактор

5.1 Цель работы

Углубить навыки по обработке сообщений мыши, расширить знания о GDI. Использование на практике шейдинга через битмап или метафайл.

5.2 Предварительные сведения

Как было сказано ранее, вывод графической информации в Windows осуществляется при помощи функций подсистемы GDI. Когда программе необходимо начать рисование, она должна получить дескриптор контекста устройства. Например, используя функции `BeginPaint` или `GetDC`. После окончания рисования программа должна освободить дескриптор. Между этими действиями можно осуществлять вывод на контекст устройства. Типы графических объектов, выводимых на контекст устройства, которые могут быть разделены на несколько категорий, часто называют "примитивами". Это:

- 1 Прямые (отрезки) и кривые. Прямые — основа любой векторной графической системы. GDI поддерживает прямые линии, прямоугольники, эллипсы (включая окружности), дуги, являющиеся частью кривой эллипса, и сплайны Безье. Более сложные кривые могут быть изображены как ломаные линии, которые состоят из очень коротких прямых, определяющих кривые. Линии рисуются с использованием пера, выбранного в контексте устройства.
- 2 Закрашенные области. Если набор прямых и кривых линий ограничивает со всех сторон некоторую область, то она может быть закрашена с использованием объекта GDI "кисть", выбранного в контексте устройства. Эта кисть может быть

сплошной, штриховой (состоящей из горизонтальных, вертикальных или диагональных штрихов) или шаблонной, заполняющей область горизонтально и вертикально.

- 3 Битовые шаблоны (растровые шаблоны, растровые образы). Битовые шаблоны — это двумерный массив битов, соответствующий пикселям устройства отображения. Это базовый инструмент в растровой графике. Битовые образы используются, в основном, для отображения сложных (часто из реального мира) изображений на дисплее или принтере. Битовые образы также используются для отображения маленьких картинок, таких как значки, курсоры мыши, кнопки панели инструментов программ, которые нужно быстро нарисовать.
- 4 Текст. Текст отличается от других математических объектов компьютерной графики. Типов текста бесконечно много. Это известно из многолетней истории типографского дела, которое многие считают искусством. Поэтому поддержка текста часто наиболее сложная часть в системах компьютерной графики, и, вместе с тем, наиболее важная. Структуры данных, используемые для описания объектов GDI — шрифтов, а также для получения информации о них — самые большие среди других структур данных в Windows.

Как и в случае создания собственных шрифтов, имеется также возможность создания собственных перьев и кистей. Схема при этом остаётся такой же, как и в случае использования собственных шрифтов.

- 1 Создаётся объект. Перо можно создать при помощи функции `CreatePen`, а кисть — при помощи `CreateBrushIndirect`.
- 2 Заменяется старый объект на новый при помощи функции `SelectObject`. При этом старый объект запоминается.
- 3 Рисуются примитивы с использованием нового объекта.

- 4 При помощи функции `SelectObject` возвращается старый объект на контекст устройства.
- 5 При помощи функции `DeleteObject` удаляется созданный объект.

5.2.1 Битовые образы

Windows содержит следующие функции, которые позволяют вам в программе создать зависящий от устройства битовый образ — объект GDI.

```
hBitmap = CreateBitmap(cxWidth, cyHeight, iPlanes,
iBitsPixel, pBits);
hBitmap = CreateBitmapIndirect(&bitmap);
hBitmap = CreateCompatibleBitmap(hdc, cxWidth,
cyHeight);
hBitmap = CreateDiscardableBitmap(hdc, cxWidth,
cyHeight);
```

Во всех случаях параметры `cxWidth` и `cyHeight` — это ширина и высота битового образа в пикселях. В функции `CreateBitmap` параметры `iPlanes` и `iBitsPixel` — это число цветовых плоскостей и число битов цвета на пиксель в битовом образе. Хотя бы один из этих двух параметров должен быть равен 1. Если оба параметра равны 1, то функция строит монохромный битовый образ.

В функции `CreateBitmap` параметр `pBits` может быть установлен в `NULL`, если вы создаете неинициализированный битовый образ. Созданный битовый образ будет содержать случайные данные. В функциях `CreateCompatibleBitmap` и `CreateDiscardableBitmap` Windows использует контекст устройства, описываемый параметром `hdc`, для получения числа цветовых плоскостей

и числа битов цвета на пиксель. Битовый образ, создаваемый этими функциями, будет неинициализированным. Функция `CreateBitmapIndirect` схожа с функцией `CreateBitmap` за исключением того, что она использует структуру типа `BITMAP` для задания битового образа.

После создания битового образа вы уже не можете изменить размер, число цветовых плоскостей или число битов цвета на пиксель. Вам надо будет создать новый битовый образ и передать биты из исходного битового образа в новый. Если вы имеете дескриптор битового образа, вы можете узнать его размер и цветовую организацию следующим образом:

```
GetObject(hBitmap, sizeof(BITMAP), (LPVOID) &bitmap);
```

Эта функция копирует информацию о битовом образе в структуру (с именем `bitmap`) типа `BITMAP`. Эта функция не устанавливает поле `bmBits`. Для доступа к битам битового образа вам нужно вызвать функцию:

```
GetBitmapBits(hBitmap, dwCount, pBits);
```

Она копирует `dwCount` бит в символьный массив по указателю `pBits`. Для того, чтобы быть уверенными, что все биты битового образа скопируются в этот массив, вы можете вычислить параметр `dwCount` на основе значений полей структуры битового образа:

```
dwCount = (DWORD) bitmap.bmWidthBytes *  
bitmap.bmHeight * bitmap.bmPlanes;
```

Вы можете также задать Windows скопировать символьный массив, содержащий биты битового образа, обратно в существующий битовый образ, используя функцию:


```
SetBitmapBits(hBitmap, dwCount, pBits);
```

Поскольку битовые образы — это объекты GDI, вы должны удалить каждый созданный вами битовый образ:

```
DeleteObject(hBitmap);
```

Контекст памяти (memory device context) — это контекст, имеющий поверхность отображения (display surface), существующую только в памяти. Вы можете создать контекст памяти, используя функцию `CreateCompatibleDC`:

```
hdcMem = CreateCompatibleDC(hdc);
```

Дескриптор `hdc` — дескриптор действительного открытого контекста устройства. Функция `CreateCompatibleDC` возвращает дескриптор контекста памяти. При создании контекста памяти все атрибуты устанавливаются в значения по умолчанию. Вы можете делать почти все, что вы захотите с этим контекстом памяти. Вы можете устанавливать атрибуты в значения, отличные от значений по умолчанию, получать текущие значения атрибутов, выбирать в него перья, кисти и регионы. И, конечно, вы можете даже рисовать на нем.

Для больших и сложных манипуляций с пикселями в Windows есть функции `BitBlt`, `PatBlt` и `StretchBlt`. `BitBlt` означает перенос блоков битов. Функция `BitBlt` переносит пиксели, другими словами, это — универсальная растровая функция.

```
BitBlt(hdcDest, xDest, yDest, xWidth, yHeight,  
hdcSrc, xSrc, ySrc, dwROP);
```

где: `hdcDest` — дескриптор того контекста устройства, на который переносится изображение;

`xDest`, `yDest` — координаты, куда будет переноситься изображение;

`xWidth`, `yHeight` — ширина и высота переносимого изображения;

`hdcSrc` — дескриптор того контекста устройства, с которого переносится изображение;

`xSrc`, `ySrc` — координаты левого верхнего угла, откуда переносится изображение;

`dwROP` — константа, означающая, каким образом изображение будет копироваться. Для обычного копирования необходимо использовать флаг `SRCCOPY`.

5.3 Задания

- 1 Разработать растровый графический редактор, состоящий из панели инструментов и области рисования.
- 2 Обеспечить возможность изменения цвета и размера пера и кисти.
- 3 Добавить инструмент «Линия».
- 4 Добавить инструмент, отвечающий за прорисовку объекта, указанного в таблице 5.1.

Таблица 5.1 - Задания на лабораторную работу № 5

Номер варианта	Ресурс
1	Треугольник
2	Прямоугольник
3	Эллипс
4	Пятиугольник

6 Лабораторная работа №7. Динамически загружаемые библиотеки (DLL)

6.1 Цель работы

Научиться разрабатывать и использовать динамически загружаемые библиотеки (DLL).

6.2 Предварительные сведения

Динамически подключаемые библиотеки (DLL, или динамические библиотеки, или библиотеки динамической компоновки, или модули библиотек) являются одним из наиболее важных структурных элементов Windows. Большинство файлов, из которых состоит Windows, представляют из себя либо программные модули, либо модули динамически подключаемых библиотек.

Как известно, Windows-программа представляет собой исполняемый файл, который обычно создает одно или более окон, а для получения данных от пользователя использует цикл обработки сообщений. Динамически подключаемые библиотеки, как правило, непосредственно не выполняются и обычно не получают сообщений. Они представляют из себя отдельные файлы с функциями, которые вызываются программами и другими динамическими библиотеками для выполнения определенных задач. Динамически подключаемая библиотека активизируется только тогда, когда другой модуль вызывает одну из функций, находящихся в библиотеке. Хотя модуль динамически подключаемой библиотеки может иметь любое расширение (например, .EXE или .FON), стандартным расширением, принятым в Windows, является .DLL.

Разберём пример использования динамической библиотеки. В библиотеке «Helloword» находится единственная функция

ShowHelloWorld, которая при помощи функции MessageBox выводит сообщение «Hello World». Такая библиотека состоит из модуля *helloworld.c*

```
#include <windows.h>
#define DLL_COMPILE
#include "helloworld.h"
int WINAPI DllMain(HINSTANCE hInstance, DWORD
fdwReason, PVOID pvReserved) {
return TRUE;
}

BOOL EXPORT ShowHelloWorld(HWND hwnd) {
    MessageBox(hwnd, "Hello, World", "Dll library",
MB_OK | MB_ICONINFORMATION);
    return TRUE;
}
```

С заголовочным файлом *helloworld.h*

```
#ifndef DLL_COMPILE
#define EXPORT __export
#else
#define EXPORT __import
#endif //DLL_COMPILE

BOOL EXPORT ShowHelloWorld(HWND);

#ifdef __cplusplus
extern "C" {
#endif // __cplusplus
```

```
typedef BOOL (WINAPI *TShowHelloWorld)(HWND hwnd);
#ifdef __cplusplus
    }
#endif // __cplusplus
```

Если перед в описании функции стоит ключевое слово `__export`, то это означает, что данная функция может использоваться извне библиотеки, если же в декларации функции стоит слово `__import`, то это означает, что данная функция будет подключена извне библиотеки. Объявив вышеописанным способом макрос `EXPORT`, мы добились того, что функция `ShowHelloWorld` будет импортируемая или экспортируемая в зависимости от того, объявлен ли макрос `DLL_COMPILE`. В заголовочном файле мы также объявили тип «указатель на функцию типа `ShowHelloWorld`», который нам понадобится при подключении библиотеки. Для использования библиотеки составим модуль *testprogramm.c*

```
#include <windows.h>
#include "helloworld.h"
int WINAPI DllMain(HINSTANCE hInstance, DWORD
fdwReason, PVOID pvReserved) {
    HINSTANCE hlib = LoadLibrary("helloworld.dll");
    TShowHelloWorld fShowHelloWorld =
(TShowHelloWorld)GetProcAddress(hlib,
"_ShowHelloWorld");
    fShowHelloWorld(NULL);
    FreeLibrary(hlib);
}
```

Сперва, при помощи функции `LoadLibrary` мы загружаем в память dll-библиотеку, затем при помощи функции `GetProcAddress` получаем адрес функции `ShowHelloWorld`. Здесь нам и понадобился тип «указатель на функцию `TShowHelloWorld`». После завершения использования библиотеки, её необходимо выгрузить при помощи функции `FreeLibrary`.

6.3 Задания

- 1 Разработать DLL-библиотеку, содержащую функции для работы с объектами, описанными в таблице 6.1. Доступ к объектам должен осуществляться через дескрипторы. Не допускается получать прямой доступ к внутреннему представлению объектов.
- 2 Разработать программу, содержащую графический интерфейс для тестирования функций разработанной DLL-библиотеки. Библиотеку подгружать по технологии «Run on call» при помощи функций `LoadLibrary`, `GetProcAddress` и `FreeLibrary`.

Таблица 6.1 — Задания на лабораторную работу № 6

Номер варианта	Объекты	Функции
1	Односвязанный список, хранящий целые числа.	Создать список. Функция должна вернуть дескриптор списка.
		Добавить элемент в конец списка. Функция должна вернуть дескриптор элемента списка.
		Добавить элемент в начало списка. Функция должна вернуть дескриптор элемента списка.
		Получить дескриптор начального элемента списка.
		Перейти на следующий элемент списка. На входе функции подаётся дескриптор элемента, следующий элемент от которого требуется получить.
		Получить значение элемента списка по его дескриптору.
		Перебрать список. На входе функции подаётся callback функция с параметрами «Дескриптор списка» и «Дескриптор элемента списка». Callback функция должна вызываться для каждого элемента списка.
		Очистить список.
		Удалить список.

Продолжение таблицы 6.1

Номер варианта	Объекты	Функции
2	Бинарное дерево, хранящее целые числа.	Создать дерево вместе с корневым элементом. Функция должна вернуть дескриптор дерева и дескриптор корневого элемента.
		Получить корневой элемент дерева. Функция должна вернуть дескриптор корневого элемента.
		Получить подэлемент справа. На вход функции подаётся дескриптор элемента, на выходе выдаётся подэлемент справа.
		Получить подэлемент слева. На вход функции подаётся дескриптор элемента, на выходе выдаётся подэлемент слева.
		Получить значение элемента дерева по его дескриптору.
		Добавить подэлемент справа. На вход функции подаётся дескриптор элемента, к которому добавляется подэлемент и значение подэлемента.
		Добавить подэлемент слева. На вход функции подаётся дескриптор элемента, к которому добавляется подэлемент и значение подэлемента.
		Очистить дерево.
		Удалить дерево.

Продолжение таблицы 6.1

Номер варианта	Объекты	Функции
3	Динамический массив, хранящий целые числа.	Создать динамический массив. На выходе функция должна возвращать дескриптор массива.
		Установить длину динамического массива.
		Получить длину массива.
		Задать значение элемента массива. На входе функция должна получать дескриптор массива, номер элемента и значение этого элемента. Функция должна проверять, не выходит ли номер элемента за пределы массива.
		Получить значение элемента массива. На входе функция должна получать дескриптор массива и номер элемента. Функция должна проверять, не выходит ли номер элемента за пределы массива.
		Удалить массив.

Окончание таблицы 6.1

Номер варианта	Объекты	Функции
4	Двусвязанный список, хранящий целые числа.	Создать список. Функция должна вернуть дескриптор списка.
		Добавить элемент в конец списка. Функция должна вернуть дескриптор элемента списка.
		Добавить элемент в начало списка. Функция должна вернуть дескриптор элемента списка.
		Получить дескриптор начального элемента списка.
		Получить дескриптор последнего элемента списка.
		Перейти на следующий элемент списка. На входе функции подаётся дескриптор элемента, следующий элемент от которого требуется получить.
		Перейти на предыдущий элемент списка. На входе функции подаётся дескриптор элемента, следующий элемент от которого требуется получить.
		Очистить список.
		Удалить список.

Приложение А

Пример оформления отчёта по лабораторной работе № 1

Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И
РАДИОЭЛЕКТРОНИКИ
(ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

Отчет по лабораторной работе №1 по предмету
«Системное программное обеспечение»

М.А. _____

г.

каф.КСУП

М.А. _____

г.

Выполнил студент
группы 580-1
Песков

«__» _____ 2006

Принял
Ассистент

Песков

«__» _____ 2006

Введение

[Здесь нужно написать, что такое Windows-программа]

Структура Windows-программы

[Здесь нужно написать про составляющие Windows-программы: регистрацию класса окна, создание окна, цикл обработки сообщений, оконную процедуру]

Стили окна

[Здесь нужно написать о том, что такое стили окна и дескриптор, как выглядит окно со стилем, указанным в вашем варианте]

Выводы

[Здесь нужно сделать выводы о проделанной работе]