Лабораторная работа №3

Изучение механизмов ввода информации от клавиатуры, мыши и таймера

Работа выполняется в системе программирования Borland Delphi 2.0 в среде Windows 95. Использование средств визуального программирования и классов VCL не допускается, то есть Delphi используется просто как 32-разрядный компилятор с языка Паскаль для операционной системы Windows.

Теоретические положения

Операционная система Windows извещает запущенные процессы о событиях, в том числе о событиях от устройств ввода, путем посылки им сообщений. Сообщения от клавиатуры, мыши и таймера помещаются в очереди сообщений потоков, откуда извлекаются потоками-получателями и отправляются на обработку в оконные процедуры. Логика прохождения сообщений от клавиатуры, мыши и таймера заслуживает отдельного описания.

Клавиатура

Сообщения от клавиатуры можно разделить на аппаратные и символьные.

Аппаратные сообщения возникают при нажатии и отпускании клавиш на клавиатуре и содержат аппаратно-независимый виртуальный код нажатой клавиши Windows, а также скэн-код. Аппаратные сообщения помещаются в очередь сообщений процесса операционной системой.

Символьные сообщения содержат код символа (или управляющего символа), если была нажата алфавитно-цифровая клавиша. Символ зависит не только от нажатой клавиши, но и от положения клавиш Shift, CapsLock, установленного языкового драйвера. Символьные сообщения помещаются в очередь при обработке аппаратных сообщений о нажатии клавиш процедурой TranslateMessage, которая вызывается по инициативе самого процесса в цикле обработки сообщений:

```
while GetMessage(msg,0,0,0) do begin {получить очередное сообщение}
  TranslateMessage(msg);   {Windows транслирует сообщения от клавиатуры}
  DispatchMessage(msg);    {Windows вызовет оконную процедуру}
end; {выход по wm_quit, на которое GetMessage вернет FALSE}
```

Аппаратные сообщения клавиатуры

Аппаратные сообщения от клавиатуры накапливаются в общесистемной очереди, откуда посылаются в очередь сообщений окна, имеющего фокус ввода. Пока окно не обработает сообщение, следующее сообщение не посылается. Это связано с тем, что обработка сообщения от клавиатуры может приводить к смене окна, имеющего фокус ввода, и последующие сообщения должны быть адресованы уже новому окну. Поэтому, если система по каким-либо причинам не может осуществить обработку сообщений от клавиатуры в темпе их поступления, наличие общесистемной очереди клавиатурных сообщений оказывается оправданным. (Окно извещается о получении или потере фокуса ввода посылкой ему сообщений WM_SETFOCUS и WM_KILLFOCUS.)

При нажатии клавиши окну посылается сообщение WM_KEYDOWN или WM_SYSKEYDOWN, при отпускании — WM_KEYUP или WM_SYSKEYUP. При длительном удержании клавиши нажатой сообщения о нажатии клавиши посылаются

многократно. Сообщения, в идентификаторах которых фигурирует слово "SYS", свидетельствуют о том, что клавиша нажимается при нажатой клавише Alt.

Все перечисленные аппаратные сообщения от клавиатуры в параметре WPARAM имеют виртуальный код клавиши Windows. Эти коды имеют идентификаторы, начинающиеся с VK_ , и их можно найти в файле WINDOWS.PAS.

В LPARAM содержится дополнительная информация о нажатии клавиши, структура которой такова:

Биты	Размер	Значение
0..15	16	Счетчик повторений
16..23	8	Скэн-код
24	1	Флаг принадлежности расширенной клавиатуре
29	1	Флаг контекста. "0" для обычных и "1" для системных сообщений.
30	1	Предыдущее состояние клавиши. "1" - была нажата, "0" - была отпущена.
31	1	Состояние клавиши. "1" - нажата (KEYDOWN), "0" - отпущена (KEYUP).

Счетчик повторений содержит количество повторов нажатия клавиши, которое может отличаться от 1 в случае, когда пользователь нажимает и удерживает клавишу, а программа не в состоянии обработать сообщения клавиатуры в темпе их поступления. В такой ситуации Windows не помещает в очередь отдельные сообщения, а накапливает нажатия в одном единственном сообщении, увеличивая поле счетчика в параметре LPARAM.

Скэн-код клавиши предоставляется в соответствующем поле, однако программа для идентификации клавиши должна использовать ее виртуальный код, передаваемый в WPARAM.

Флаг принадлежности к расширенной клавиатуре выставляется в единицу, если нажаты такие клавиши, как правый Alt, Ctrl или Shift, "серые" клавиши со знаками арифметических операций и Enter, NumLock, а также клавиши управления курсором, не относящиеся к клавиатуре цифрового набора.

Флаг контекста показывает, нажата ли клавиша Alt в момент посылки сообщения. Когда активное окно минимизировано, оно получает все клавиатурные сообщения как системные, и чтобы определить, нажата ли действительно клавиша Alt, используется данный флаг.

Как правило, пользовательский процесс не должен реагировать на системные сообщения клавиатуры, передавая их в DefWindowProc. Чаще всего программа должна обрабатывать только сообщения WM_KEYDOWN от клавиш управления курсором.

Символьные сообщения клавиатуры

После обработки аппаратного сообщения в TranslateMessage в очередь могут быть помещены символьные сообщения WM_CHAR, WM_SYSCHAR, WM_DEADCHAR и WM_SYSDEADCHAR. Сообщения со словом "SYS" являются следствием сообщений WM_SYSKEYDOWN, без него — WM_KEYDOWN.

Символьные сообщения содержат код нажатого символа в WPARAM, а в LPARAM содержится та же информация, что и у породившего их аппаратного сообщения, в том числе — количество повторов символа.

DEADCHAR (немой символ) имеет смысл для некоторых европейских раскладок клавиатуры, когда символ с диакритическим знаком вводится путем последовательного нажатия клавиши соответствующего знака (например — апострофа) и алфавитной клавиши. В результате окно получает два символьных сообщения: WM_DEADCHAR с символом диакритического знака и, при последующем нажатии алфавитной клавиши, WM_CHAR с символом с диакритическим знаком, например символом Ó . Если введенный после немого символа алфавитный символ не может иметь диакритического знака, окно получит не два, а три сообщения: WM_DEADCHAR и WM_CHAR с диакритическим знаком и WM_CHAR с буквой алфавита. Очевидно, что поскольку логика обработки немых символов реализована в Windows, пользовательской программе обычно нет необходимости обрабатывать сообщения о немых символах.

Состояние клавиатуры

Windows содержит ряд функций для опроса и переключения состояния клавиатуры. Эти функции возвращают не текущее состояние клавиатуры в момент их вызова, а состояние клавиатуры на момент посылки последнего извлеченного из очереди клавиатурного сообщения. Таким образом можно считать, что к каждому сообщению о нажатии клавиши "привязана" информация о состоянии всех клавиш в этот момент, и именно с этой информацией о "логическом состоянии клавиш" можно производить операции.

function GetKeyState(nVirtKey: Integer): SmallInt;

Функция GetKeyState возвращает логическое состояние клавиши с соответствующим виртуальным кодом. Старший бит 16-разрядного результата взведен, если клавиша нажата, и сброшен, если клавиша отпущена. Младший бит для клавиш управления режимом (CapsLock, NumLock, ScrollLock) показывает, активен соответствующий режим ("1") или нет ("0").

Следующие две функции позволяют прочитать и модифицировать логическое состояние клавиатуры целиком. Возвращаемое логическое значение свидетельствует об успехе или неуспехе производимой операции.

type TKeyboardState = **array**[0..255] **of** Byte;

function GetKeyboardState(**var** KeyState: TKeyboardState): BOOL;

function SetKeyboardState(**var** KeyState: TKeyboardState): BOOL;

Старший бит соответствующего виртуальному коду клавиши байта взведен, когда клавиша нажата, и сброшен, когда отпущена. Младший бит по-прежнему показывает режим для клавиш управления режимом.

Для определения физического, т.е. аппаратного состояния клавиши в текущий момент служит функция GetAsyncKeyState с параметрами, аналогичными GetKeyState.

Мышь

Использование указательного устройства в графических средах, подобных Windows, является стандартом. В качестве такового в Windows используется устройство типа "мышь" с одной, двумя или тремя кнопками. Для Windows 95 стандартной является двухкнопочная мышь.

Функция GetSystemMetrics позволяет определить следующие параметры:

getSystemMetrics(SM_MOUSEPRESENT) — не 0, если в системе установлена мышь

getSystemMetrics(SM_CMOUSEBUTTONS) — количество кнопок мыши

getSystemMetrics(SM_SWAPBUTTON) — не 0, если включено зеркальное отображение кнопок мыши (правая воспринимается как левая и наоборот).

С событиями от мыши связано 21 сообщение Windows. Если курсор мыши находится в рабочей области окна, окно получает следующие сообщения:

Движение курсора: WM_MOUSEMOVE

Нажатие кнопок:

Кнопка	**Нажатие**	**Отпускание**	**Двойное (второе) нажатие***
Левая	WM_LBUTTONDOWN	WM_LBUTTONUP	WM_LBUTTONDBLCLK
Правая	WM_RBUTTONDOWN	WM_RBUTTONUP	WM_RBUTTONDBLCLK
Средняя	WM_MBUTTONDOWN	WM_MBUTTONUP	WM_MBUTTONDBLCLK

* Сообщения о двойных щелчках окна получают лишь в том случае, когда в стиле их оконного класса указан флаг CS_DBLCLKS. В противном случае для отслеживания многократных щелчков в оконной процедуре удобно использовать функцию GetMessageTime для определения моментов посылки сообщений и интервалов между ними.

Для всех этих сообщений параметры следующие:

wParam - состояние кнопок, а также клавиш Ctrl и Shift клавиатуры; OR-комбинация констант MK_LBUTTON, MK_RBUTTON, MK_MBUTTON, MK_SHIFT и MK_CONTROL.

lParam - координаты курсора в пикселах относительно левого верхнего угла рабочей области, X=loWord(lParam); Y=hiWord(lParam);

В отличие от сообщений клавиатуры, получение сообщений от мыши определяется не активностью окна, а положением курсора мыши в области окна. Сообщения о нажатии кнопок мыши могут не сопровождаться соответствующими сообщениями об их отпускании, если курсор вдруг будет выведен за границы окна.

Windows помещает сообщения от мыши в очередь сообщений процесса только в том случае, если там нет однотипных сообщений. Из-за этого в случае перегруженности системы возможно "пропадание" некоторых действий с мышью. Например, если пользователь щелкнул кнопкой мыши в поле окна, то соответствующие сообщения помещаются в очередь, но пока они не будут извлечены оттуда программой, следующие нажатия и отпускания кнопки мыши в этом окне будут пропадать, т.е. очередные сообщения WM_xBUTTONDOWN и WM_xBUTTONUP в очередь помещены не будут.

Захват мыши

Часто окну бывает необходимо получать сообщения от мыши даже в том случае, если курсор мыши покинет пределы рабочей области. Например, если пользователь нажимает кнопку мыши для перетаскивания какого либо объекта в окне графического редактора, это окно всегда должно знать, когда кнопка мыши будет отпущена, чтобы завершить или отменить операцию перетаскивания. Для этого в обработчике сообщения WM_LBUTTONDOWN окном должен быть выполнен захват мыши, а в WM_LBUTTONUP — освобождение захвата.

Захват мыши осуществляется функцией SetCapture(hWnd), которая возвращает нулевое значение или хэндл окна, осуществлявшего захват мыши до вызова этой функции. Освобождение захвата осуществляется вызовом функции ReleaseCapture без параметров. Окно, захватившее мышь, получает сообщения от мыши, когда курсор находится вне его,

только в том случае, если кнопка мыши нажата. В противном случае сообщения мыши проходят стандартным образом, невзирая на захват.

Сообщения мыши нерабочей области

Окно получает извещение о действиях с мышью и тогда, когда курсор находится на нерабочей области окна — на заголовке или на рамке. Названия этих сообщений имеют в своем составе аббревиатуру NC (non-client), и трактовка их параметров отличается от трактовки параметров сообщений для рабочей области.

Это сообщения:

WM_NCMOUSEMOVE, WM_NCLBUTTONDOWN, WM_NCRBUTTONDOWN, WM_NCLMUTTONDOWN, WM_NCLBUTTONUP, WM_NCRBUTTONUP, WM_NCLMUTTONUP, WM_NCLBUTTONDBLCLK, WM_NCRBUTTONDBLCLK, WM_NCLMUTTONDBLCLK.

wParam - обозначает зону окна. Полный список возможных вариантов можно получить в описании сообщения WM_NCHITTEST.

lParam - как и для сообщений рабочей области — координаты курсора, но в координатной системе всего экрана, а не рабочей области окна.

Преобразование координат между клиентской и экранной системами координат осуществляется функциями

function ClientToScreen(hWnd: HWND; **var** lpPoint: TPoint): BOOL;
function ScreenToClient(hWnd: HWND; **var** lpPoint: TPoint): BOOL;

Сообщение WM_NCHITTEST

Прежде чем программа получит любое сообщение от мыши, Windows запрашивает окно о том, к какой его области относится положение курсора мыши. Для этого при возникновении каждого системного события от мыши Windows посылает соответствующему окну сообщение WM_NCHITTEST, в ответ на которое оконная процедура должна вернуть значение, определяющее, где в данном окне (в рабочей области, на рамке, на заголовке и т.д.) находится указанное Windows положение курсора. Положение курсора задается в параметре lParam так же, как и в остальных сообщениях от мыши; параметр wParam не используется. В зависимости от результатов обработки сообщения WM_NCHITTEST будут сгенерированы дальнейшие сообщения для рабочей или нерабочей области окна.

Обработка этого сообщения внутри оконной процедуры пользовательского процесса открывает ряд интересных возможностей. Например, следующий фрагмент позволяет создать окно, которое можно перетаскивать по экрану не только "ухватываясь за заголовок", но и "за рабочую область":

```
case Msg of

  .....
  wm_nchittest: begin
    result:=DefWindowProc(hwnd,msg,wparam,lparam);
      {Если попали в рабочую область, то обманываем Windows
       и говорим, что в заголовок}
    if result=HTCLIENT then result:=HTCAPTION;
  end;
  .....
end;
```

Чаще всего пользовательская программа обрабатывает:
HTCAPTION - строка заголовка окна
HTCLIENT - рабочая область
HTMENU - меню
HTNOWHERE - вне окна на свободном поле экрана
HTREDUCE - кнопка минимизации окна
HTSYSMENU - значок системного меню в левом верхнем углу окна
HTZOOM - кнопка максимизации окна

Таймер

Программа Windows может запросить операционную систему, чтобы та ставила ее в известность об истечении заданных промежутков времени. При этом операционная система создает логический таймер с заданным периодом, а программа начинает получать сообщения WM_TIMER. Следует заметить, что использование логического таймера не гарантирует точного отслеживания заданных промежутков времени; эти сообщения следует использовать для грубого задания интервалов времени в несколько секунд, а также для инициации периодического обновления отображаемой информации о выполняемом процессе, когда точность не важна, лишь бы информация менялась "примерно раз в N секунд".

Сообщения WM_TIMER имеют, подобно сообщениям WM_PAINT, низкий приоритет. Это означает, что сообщение WM_TIMER может быть извлечено из очереди только тогда, когда в очереди нет других сообщений. Если процесс не успел обработать предыдущее сообщение WM_TIMER, находящееся в очереди, то следующее сообщение WM_TIMER не ставится в очередь, и событие от таймера просто теряется.

Период логического таймера задается в миллисекундах и может лежать в интервале от 1 до $FFFFFFFF (что соответствует приблизительно 50 суткам). Однако для отсчета периода логических таймеров Windows использует аппаратный таймер компьютера, который в компьютерах IBM PC AT срабатывает 18.2 раза в секунду, т.е. с периодом приблизительно в 55 мсек. Из этого следует, во-первых, что задание периода таймера менее 55 мсек. бессмысленно, так как программа в этом случае все равно будет получать одно сообщение таймера каждые 55 мсек. Во-вторых, задаваемый период логического таймера всегда округляется вниз до ближайшего значения, соответствующего целому количеству срабатываний аппаратного таймера. Так, задав интервал в 1000 мсек., пользовательский процесс будет получать сообщения WM_TIMER каждые 989 мсек., точнее, с таким интервалом они будут попадать в его очередь сообщений, извлечь их оттуда вовремя — его собственная проблема.

Логический таймер создается и активизируется функцией SetTimer, а уничтожается (освобождается) функцией KillTimer:

```
function SetTimer(hWnd: THandle; nIDEvent, uElapse: integer; lpTimerFunc: pointer): integer;
function KillTimer(hWnd: THandle; uIDEvent: integer): BOOL;
```

Здесь hWnd - хэндл окна, создающего таймер, nIDEvent - числовой идентификатор таймера в программе, uElapse - период таймера в миллисекундах, lpTimerFunc - необязательный указатель на процедуру реакции на события от таймера (может быть nil).

SetTimer возвращает числовой идентификатор таймера или 0, если по каким-либо причинам таймер не может быть создан. В предыдущих версиях Windows количество логических таймеров в системе было ограничено (16, затем 32). В Windows 95 эти ограничения сняты, и невозможность создания таймера перестала быть обычной ситуацией.

Сообщение WM_TIMER имеет следующие параметры:

wParam - числовой идентификатор таймера.

lParam - указатель на функцию обработки, если таковая имеется.

Windows предоставляет две возможности по обработке сообщений таймера: в оконной процедуре и в специальной процедуре обработки. В первом случае в качестве lpTimerFunc в SetTimer передается nil, и сообщение WM_TIMER должно обрабатываться наравне со всеми остальными в общем операторе CASE оконной процедуры. Во втором случае в программе должна быть описана процедура со следующим набором параметров (имя — произвольное):

procedure TimerProc(hwnd:THandle; uMsg, idEvent, time: integer); **stdcall**;

Указатель на эту процедуру (@TimerProc) передается в качестве lpTimerFunc в SetTimer, и при каждом получении сообщения WM_TIMER процедура DispatchMessage будет вызывать не оконную процедуру, а процедуру TimerProc. Параметры этой процедуры — хэндл окна, тип сообщения (всегда WM_TIMER), числовой идентификатор таймера и системное время в момент получения сообщения.

Windows при использовании процедуры реакции таймера предоставляет возможность создавать таймер без окна, указав в параметре hwnd функции SetTimer значение 0. В этом случае значение параметра uIDEvent игнорируется системой, и функция возвращает номер таймера, назначенный автоматически. Этот номер должен быть впоследствии передан в KillTimer, параметр hwnd при этом также должен быть установлен в 0.

Задание

Выполняется при самостоятельной подготовке:

1. Изучить теоретическую часть описания ЛР и материал соответствующих лекций.

2. По материалам Справки (Help) изучить описание следующих сообщений Windows и связанных с ними функций и структур данных:

Сообщения: WM_KEYDOWN, WM_KEYUP, WM_CHAR, WM_DEADCHAR, WM_SYSKEYDOWN, WM_SYSKEYUP, WM_SYSCHAR, WM_NCMOUSEMOVE, WM_NCLBUTTONDOWN, WM_NCRBUTTONDOWN, WM_NCLMUTTONDOWN, WM_NCLBUTTONUP, WM_NCRBUTTONUP, WM_NCLMUTTONUP, WM_NCLBUTTONDBLCLK, WM_NCRBUTTONDBLCLK, WM_NCLMUTTONDBLCLK, WM_NCHITTEST, WM_TIMER.

Функции: TranslateMessage, GetKeyState, GetKeyboardState, SetKeyboardState, GetAsyncKeyState, Setcapture, ReleaseCapture, ClientToScreen, ScreenToClient, SetTimer, KillTimer, GetTickCount.

3. Продумать алгоритм решения индивидуального задания.

Выполняется в лаборатории:

1. Включить компьютер. Запустить систему Delphi 2.0.

2. Загрузить исходный текст примера LAB5.PAS, а также модуль WINDOWS.PAS, изучить логику работы программы.

3. Откомпилировать и запустить пример. Изучить поведение созданного окна.

4. Написать и отладить программу по индивидуальному заданию (см. ниже). Продемонстрировать результаты работы преподавателю.

5. Завершить работу в Delphi. Оставить компьютер включенным.

Варианты заданий:

1. Программа-секундомер. При нажатии клавиши Enter в рабочей области окна начинает отображаться отсчитываемое время. Точность отсчета - десятые доли секунды. При повторном нажатии Enter отсчет останавливается. Третье нажатие Enter обнуляет время. Окно секундомера — масштабируемое, без заголовка. Перемещение окна реализовать при помощи перетаскивания мышью "за клиентскую область".

Примечание: Пытаясь отслеживать время с точностью до 0.1 секунды, помните о невысокой точности сообщений таймера Windows.

2. В рабочей области окна существует единственный объект, представляющий собой геометрическую фигуру, отличную от прямоугольника (например, круг или треугольник). Нажатием цифровых клавиш 1..8 цвет объекта изменяется на один из 8 возможных. Стрелками осуществляется перемещение объекта на 1 пиксел в любую из 4 сторон. Предусмотреть также возможность перетаскивания объекта мышью.

Примечание: объект удобно представлять как регион, см. соответствующую группу функций в справке.

3. Программа - телетайп. Используя моноширинный шрифт (например, Courier) программа организует окно с размером клиентской области 80x25 символов. Алфавитно-цифровые символы выводятся на этот экран. Символ конца строки приводит к переходу в начало следующей строки, остальные управляющие символы обрабатывать не нужно. При достижении конца строки набор продолжается в следующей строке. При достижении конца экрана содержимое прокручивается на 1 строку, при этом верхняя строка теряется. При щелчке левой кнопкой мыши по какому-либо знакоместу весь экран телетайпа заполняется соответствующим символом, находящимся в указанной позиции. Окно очищается при нажатии клавиши Esc.

4. Программа - монитор состояния клавиатуры. В окне в произвольном виде отображается состояние (нажато-отпущено) клавиш на клавиатуре. Можно ограничиться только алфавитно-цифровыми клавишами. Отслеживание должно производиться даже тогда, когда окно не имеет фокуса ввода, т.е. следует асинхронно сканировать состояние клавиатуры. Окно при этом должно быть управляемым, т.е. приходящие в очередь сообщения должны обслуживаться.

5. "Уворачивающееся" окно. При попытке ввести курсор мыши в область, занятую окном (в том числе в неклиентскую область) окно изменяет свое положение таким образом, чтобы курсор мыши оказался вне окна. Окно не должно полностью уходить за границы экрана, а производимый сдвиг должен быть по возможности минимальным.

6. "Вручную", т.е. без собственной обработки WM_NCHITTEST, реализовать перетаскивание окна без заголовка "за клиентскую область". При этом желательно, чтобы в процессе перетаскивания текущее положение окна отображалось прямоугольной рамкой. Закрывать окно по нажатию клавиши Esc.

7. Прототип программы-"калькулятора", складывающей целые неотрицательные числа. В рабочей области отображается "индикатор" и 10 "клавиш" с цифрами, клавиши "Сброс(Esc)", "+" и "=". Последовательность применения калькулятора: набрать цифры первого числа (не более 8), нажать "+", набрать цифры второго числа (не более 8), нажать "=" - отобразится сумма, а калькулятор готов к приему очередного первого числа. При нажатии клавиши "Сброс" калькулятор переводится в состояние ожидания ввода первого числа, а сумма становится нулевой. Реализовать возможность ввода управляющих воздействий как с клавиатуры, так и мышью с помощью нарисованных в рабочей области клавиш.

Примечание: Удобно было бы реализовать ввод с клавиатуры, а затем при щелчке мышью по нарисованной клавише посылать окну сообщение WM_CHAR с кодом соответствующей клавиши клавиатуры. Пусть окно будет не масштабируемым.

8. Окно с изменяемым цветом фона, задаваемым в формате RGB (красный-зеленый-синий). Клавиши с буквами R, G и B ответственны за изменение каждой из составляющих цвета: если нажата клавиша Shift и "R", "G" или "B", то соответствующая составляющая цвета фона увеличивается на 1, при отпущенной клавише Shift - уменьшается. Из трех клавиш "R", "G" и "B" допускается одновременно нажимать (и удерживать) более одной, тогда должны изменяться несколько цветовых составляющих сразу. Цветовая составляющая не должна становиться меньше 0 и больше 255, т.е. выходить за диапазон BYTE. Предусмотреть вывод текущих значений составляющих цвета (допустимо использовать функцию IntToStr из модуля SysUtils в составе Delphi).

Примечание: в ответ на сообщение WM_KEYDOWN необходимо проверять состояние всех клавиш клавиатуры.

9. "Рисование" мышью. При нажатии левой кнопки мыши производится рисование точки. При движении мышью с нажатой левой кнопкой производится рисование линии по координатам, проходимым мышью. При нажатии правой кнопки производится заливка области (см. FloodFill). Цвет пера и кисти выбирается из 8 возможных при нажатии соответственно клавиш 1..8 и F1..F8.

Примечание: не требуется реализовывать запоминание рисунка и его восстановление в случае, если рабочая область окна по каким-то причинам станет недействительной.

10. Программа-"метроном". Рабочая область окна (или ее участок) должны периодически изменять цвет (например, с черного на белый и наоборот) через постоянные интервалы времени, например — каждую секунду. Допустимый интервал — от 0.5 секунды до 3 секунд. Изменение интервала с шагом в 0.1 секунду производится при нажатии клавиш "стрелка вверх" и "стрелка вниз", с шагом 0.5 сек. — при нажатии тех же клавиш в комбинации с клавишей Ctrl.