Лабораторная работа № 2

Тема: Разработка программ с использованиемWin32 API
Цель: Освоить создание приложений с графическим интерфейсом с использованием функцийWin32 API: создание и регистрация класса окна, создание и отображение окна, обработка событий.

Краткая теория

Библиотека Win32 API – библиотека, содержащая функции ОС Windows предназначенные для организации взаимодействия прикладных программ пользователя с сервисом, предоставляемым ОС Windows. В программу необходимо, как минимум, подключить библиотеку <windows.h>. При использовании специфичных сервисов возможно понадобится подключение и других библиотек.

В библиотеке Win32 API содержится описание довольно большого количества типов данных, большинство из которых создано с использованием директив #define или typedef. Эти типы данных могут переопределять системные типы данных языков C/C++ таких как int, long, char*, constchar* и д.р. Также эти типы данных могут содержать объявления структур, предназначенных для описания каких-либо сложных объектов, посредством которых осуществляется взаимодействие с операционной системой.

В таблице 1 приведены некоторые из типов данных, описанные в библиотеках Win32API.

Таблица 1 – Типы данных Win32 API

HANDLE	Дескриптор некоторого ресурса, связанного с приложением (целое 32-ух разрядное число)
HWND	Идентификатор окна (целое 32-ух разрядное число)
HDC	Идентификатор контекста устройства (целое 32-ух разрядное число)
LONG	Целое 32-разрядное число со знаком
LPSTR	Указатель на С-строку (char *)
NULL	0
INT	Целое 32-разрядное число со знаком
UINT	Целое 32-разрядное число без знака
WCHAR	16-ти разрядный символ UNICODE
BOOL	Логический тип
HBRUSH	Дескриптор кисти
HPEN	Дескриптор пера
HMENU	Дескриптор меню
LRESULT	Значение типа long, возвращаемое системными вызовами

В основе взаимодействия программы с внешним миром и с операционной системой лежит концепция сообщений. С точки зрения приложения, сообщение является уведомлением о том, что произошло некоторое событие, которое может требовать, а может и не требовать выполнения определенных действий. Это событие может быть следствием действий пользователя, например, перемещения курсора или щелчка кнопкой мыши, изменения размеров окна или выбора пункта меню. Кроме того, событие может генерироваться приложением, а также операционной системой. Сообщение – это структура данных, содержащая следующие элементы:

- дескриптор окна, которому адресовано сообщение;
- код (номер) сообщения;
- дополнительную информацию, зависящую от кода сообщения.

Сообщения в Windows описываются с помощью структуры

MSG:
```
typedefstructtagMSG {
    HWND hwnd;            // Идентификатор окна-получателя
    UINT message;         // Идентификатор сообщения
   WPARAM wParam;         // Дополнительная информация, смысл
   LPARAM lParam;         // которой зависит от типа сообщения
    DWORD time;           // Время посылки сообщения
    POINT pt;             // Местоположение указателя мыши
} MSG;
```

Сообщения от внешних источников, например, от клавиатуры, адресуются в каждый конкретный момент времени только одному из работающих приложений, а именно - активному окну. Windows играет роль диспетчера сообщений. Для этого с момента старта операционная система создает в памяти глобальный объект, называемый системной очередью сообщений. Все сообщения, генерируемые как аппаратурой, так и приложениями, помещаются в эту очередь. Windows периодически опрашивает эту очередь и, если она не пуста, посылает очередное сообщение нужному адресату, определяемому при помощи дескриптора окна. Сообщения, получаемые приложением, могут поступать асинхронно из разных источников. Например, приложение может работать с системным таймером, посылающим ему сообщения с заданным интервалом, и одновременно оно должно быть готовым в любой момент получить любое сообщение от операционной системы. Чтобы не допустить потери сообщений, Windows одновременно с запуском приложения создает глобальный объект, называемый очередью сообщений приложения.

Основу программы в Win32 составляет функция WinMain, которая «заменяет» собой стандартную функцию main языков C и C++. Описание этой функции имеет следующий вид:

```
int WINAPI  WinMain (HINSTANCE  hInstance,
        HINSTANCE  hPrevInstance, LPSTR lpCmdLine, intnCmdShow)
{ …        return Код_вовзрата;
}
```

Параметры функции WinMain:

- HINSTANCE hInstance – идентификатор текущего приложения;
- HINSTANCE hPrevInstance – идентификатор приложения, являющегося родительским для данного приложения;
- LPTSTR lpCmdLine – С-строка, содержащая параметры командной строки;
- intnCmdShow – код вида начального отображения окна.

Возвращаемое значение функции WinMain: целое число, интерпретируемое как код возврата. Обычно в качестве его значения указывается параметр сообщения закрытия приложения в виде:

```
(int) msg.wParam
```

На рисунке 1 приведен основной алгоритм реализации функции WinMain.

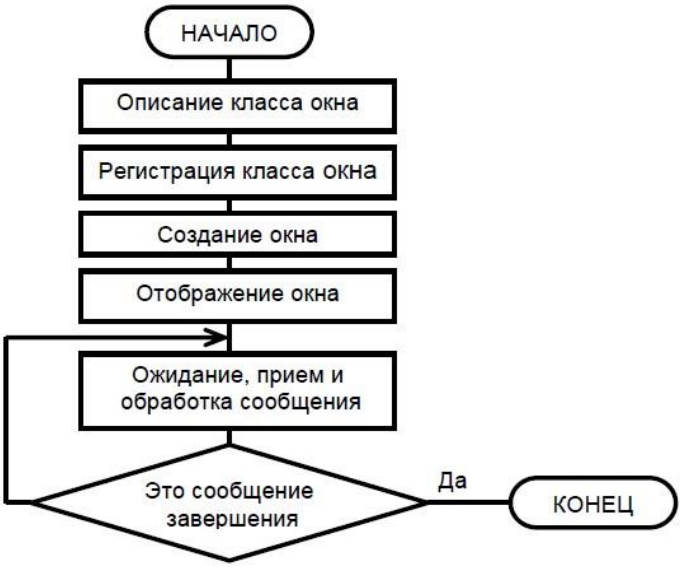

Рисунок 1 – Базовый алгоритм функции WinMain

Для описания класса окна необходимо заполнить структуру типа WNDCLASSEX или WNDCLASS (старый вариант).

```
typedefstruct tagWNDCLASSEXW {
    UINT        cbSize;       //Размер структуры
    UINT        style;        //Стиль окна
    WNDPROC      lpfnWndProc; //Адрес функции обработки сообщений
    int         cbClsExtra;   //Размер дополнительной памяти в байтах под

    int         cbWndExtra;
                              //структуру класса-окна (инициализируется 0)
                              //Размер дополнительной памяти в байтах под
                              //структуру окна (инициализируется 0)
    HINSTANCE hInstance;      //Код приложения
    HICON hIcon;

                              //Код идентификатора иконки приложения
    HCURSOR hCursor;          //Код идентификатора курсора
```

```
    HBRUSH hbrBackground;

                                //Код идентификатора кисти фона окна
    LPCWSTR lpszMenuName;

                               //Имя ресурса, описывающего меню программы
    LPCWSTR lpszClassName;    //Имя класса окна
    HICON hIconSm;            //Код идентификатора иконки класса окна
} WNDCLASSEXW;
```

После заполнения всех полей данной структуры класса окна необходимо зарегистрировать с помощью функции RegisterClassEx() или RegisterClass() (старый вариант). Прототип функции:

```
ATOM RegisterClassEx(CONST WNDCLASSEX *lpwcx );
```

В параметре в функцию передается адрес заполненной структуры типа WNDCLASSEX. Функция возвращает идентификатор зарегистрированного класса окна в случае успешного выполнения. В случае ошибки функция возвращает 0.

После успешной регистрации класса окна необходимо создать само окно. Это осуществляется с помощью функции CreateWindow. Прототип функции:

```
HWND CreateWindow(
  LPCTSTR lpClassName,        //С-строка     содержащая имя класса
  LPCTSTR lpWindowName,       //С-строка содержащая имя окна
  DWORD dwStyle,              //Стиль окна
  intx,                       //Позиция x на экране
  int y,                      //Позиция y на экране
  int nWidth,                 //Ширина окна (по осиX)
  int nHeight,                //Высота окна (по осиY)
  HWND hWndParent,            //Дескриптор родительского окна
  HMENU hMenu,                //Дескриптор главного меню
  HINSTANCE hInstance,        //Идентификатор приложения
  LPVOID lpParam              //Параметры сообщения WM_CREATE
);
```

В случае успешного выполнения функция CreateWindow возвращает дескриптор созданного окна. В случае ошибки функция возвращает NULL. Некоторые стили окна приведены в таблице 2.

Таблица 2 – Основные стили окна

WS_BORDER	Создает окно с «неподвижной» границей
WS_CAPTION	Создает окно, у которого есть заголовок
WS_CHILD	Создает дочернее окно (вложено в другое окно)
WS_DISABLED	Создает «запрещенное» окно, в него не передаются сообщения
WS_DLGFRAME	Создает диалоговое окно
WS_HSCROLL	Создает окно с горизонтальной полосой прокрутки

WS_OVERLAPPED	Создает окно с заголовком и «подвижной» границей
WS_MAXIMIZE	Создает окно изначально развернутое на весь экран
WS_MINIMIZE	Создает окно изначально свернутое
WS_POPUP	Создает «всплывающее» окно (в противоположность WS_CHILD)
WS_VISIBLE	Создает окно, изначально видимое на экране
WS_VSCROLL	Создает окно с вертикальной полосой прокрутки

После того, как окно успешно создано, его необходимо отобразить используя функции

ShowWindow и UpdateWindow.

Функция отображения окна:

```
BOOL ShowWindow(HWND hWnd, intnCmdShow);
```

Некоторые типы команд отображения окна:

- SW_HIDE – скрыть окно и активизировать другое окно,
- SW_MAXIMIZE – развернуть на весь экран,
- SW_MINIMIZE – свернуть окно и активизировать предыдущее окно,
- SW_RESTORE – восстановить изначальные размеры окна,
- SW_SHOW – активизировать окно и отобразить его. Функция возвращает истину, если окно было изначально видимо, и ложь – если нет.

Функция обновления окна:

```
voidUpdateWindow(HWNDhWnd);
```

Вызывает обновление (перерисовку) окна.

После создания и отображения окна необходимо организовать цикл получения и обработки сообщений. Для этих целей в Win32 API используются следующие функции:

- GetMessage – получение сообщения,
- TranslateMessage – преобразует сообщения виртуальных клавиш в символьные сообщения,
- DispatchMessage – вызывает обработчик сообщения.

Функция получения сообщения

```
BOOL GetMessage(
  LPMSG lpMsg,           //Указатель на структуру MSG
  HWND hWnd,             //Дескриптор окна
  UINT wMsgFilterMin,    //Минимальный номер отслеживаемых сообщений
  UINT wMsgFilterMax      //Максимальный номер отслеживаемых сообщений
);
```

Если функция получает сообщение отлично от WM_QUIT, то возвращается не нулевое значение (истина).

Функция преобразования сообщения виртуальных клавиш в символьные сообщения

```
BOOL TranslateMessage(const MSG *lpMsg );
```

Возвращает значение «истина», если сообщение было успешно преобразовано. Если сообщение не преобразовано, то возвращает значение «ложь». Данная функция предназначена для создания сообщений WM_CHAR на основе сообщений WM_KEYDOWN и WM_KEYUP.

Функция обработки сообщения

```
LRESULT DispatchMessage(const MSG *lpmsg );
```

Функция вызывает функцию-обработчик сообщений для данного окна и возвращает результат обработки. Значение результата зависит от самой функции-обработчика, и как правило игнорируется.

Типовая функция WinMain, реализующая основной алгоритм работы приложения:

```
intWINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
                  lpCmdLine, intnCmdShow)
{
     MSG msg; HWND hwnd;
     WNDCLASSEX wcx;
     //Заполнениеполейструктурыwcx

     if (!RegisterClassEx(&wcx)) return FALSE;

     hwnd = CreateWindow(/*параметры*/);

     if(!hwnd) return FALSE;

     ShowWindow(hWnd,nCmdShow);
     UpdateWindow(hWnd);

     while(GetMessage(&msg,NULL,0,0)){
          TranslateMessage(&msg);
          DispatchMessage(&msg);
     }
     return(int)msg.wParam;



}
```

Обработку сообщений, поступающих в приложение, выполняет функция-обработчик, прототип которой должен соответствовать следующему:

```
LRESULT CALLBACK WndProc(
      HWND hwnd,           //Дескриптор окна
      UINT message,        //Код сообщения
      WPARAM wParam,       //Первый параметр сообщения
      LPARAM lParam        //Второй параметр сообщения
);
```

Типовой алгоритм функции-обработчика:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message,
                         WPARAM wParam,  LPARAMlParam)
{
      switch(message){
            caseWM_DESTROY:
                  PostQuitMessage(0);
      break;      caseСОБЫТИЕ_1:
                  //Обработка события
                  1 break;
            caseСОБЫТИЕ_2:
                  //Обработка события 2
                  break;
            …
            default: return DefWindowProc(hwnd, message, wParam, lParam);
      }     return0;    }
```

В таблице 3 приведены некоторые сообщения, которые может получать приложение.

Таблица 3 – Некоторые сообщения, посылаемые приложению

WM_CREATE	Событие создания окна
WM_DESTROY	Событие уничтожения окна
WM_MOVE	Событие перемещения окна
WM_SIZE	Событие изменения размеров окна
WM_PAINT	Событие перерисовки содержимого окна
WM_COMMAND	Событие поступления сообщения для (от) элемента управления
WM_MOUSEMOVE	Событие перемещения курсора мыши
WM_LBUTTONDOWN	Событие нажатия левой кнопки мыши
WM_LBUTTONUP	Событие отпускания левой кнопки мыши
WM_LBUTTONDBLCLK	Событие двойного нажатия левой кнопки мыши
WM_RBUTTONDOWN	Событие нажатия правой кнопки мыши
WM_RBUTTONUP	Событие отпускания правой кнопки мыши
WM_RBUTTONDBLCLK	Событие двойного нажатия правой кнопки мыши
WM_KEYDOWN	Событие нажатия клавиши на клавиатуре
WM_KEYUP	Событие отпускания клавиши на клавиатуре
WM_CHAR	Событие ввода символа
WM_TIMER	Событие срабатывания таймера

Далее приведены некоторые функции для работы с окнами:

Функция определения существования окна

```
BOOL IsWindow(HWND hwnd);
```

Функция проверки наличия фокуса ввода у окна

```
BOOL IsWindowEnabled(HWND hwnd);
```

Функция разрешения или запрета фокуса ввода у окна

```
BOOL EnableWindow(HWND hwnd, BOOL flag);
```

Функция передачи фокуса управления окну

```
HWND SetFocus(HWND hwnd);
```

Функция поиска окна с заданными классом и названием

```
HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName);
```

Функция перемещения окна

```
BOOL MoveWindow(HWND hwnd, int x, int y, intnWidth,
                intnHeight, BOOL bRepaint);
```

Функция относительного перемещения окна:

```
BOOL SetWindowPos(HWND hwnd, HWND hwndInsertAfter, int x, int y,
                  intnWidth,intnHeight, UINT uFlags);
```

hwndInsertAfter может быть дескриптором существующего окна или одним из следующих значений:

- HWND_BOTTOM – помещает окно ниже других окон,
- HWND_NOTOPMOST – помещает временное или дочернее окно выше временных или дочерних окон, но ниже перекрывающихся окон,
- HWND_TOP – помещает окно выше всех окон,
- HWND_TOPMOST – то же, что HWND_NOTOPMOST, но окно сохраняет позицию после потери активности.

Вывод окна на передний план и передача ему управления:

```
BOOL SetForegroundWindow(HWND hwnd);
```

Функция получения системных метрик:

```
intGetSystemMetric(intnIndex);
```

nIndex может принимать следующие значения:

- SM_CXMIN – минимальная ширина окна,
- SM_CYMIN – минимальная высота окна,
- SM_CXSCREEN – ширина окна,
- SM_CYSCREEN – высота окна,
- SM_CYCAPTOIN – высота заголовка окна,
- SM_CYMENU – высота меню окна.

Функция получения параметров окна:

```
BOOL GetWindowRect(HWND hwnd, LPRECT rect);
```

Функция получения параметров рабочей области окна:

```
BOOL GetClientRect(HWND hwnd, LPRECT rect); typedefstruct{


 LONG left;          //Левый край
 LONG top;           //Верхний край
 LONG right;         //Правый край
 LONG bottom;        //Нижний край

} RECT;
```

Вызов окна сообщения осуществляется с помощью функции:

```
intWINAPI MessageBox(
 HWND hwnd,               //Дескриптор окна
 LPCTSTR lpText,          //C-строка текста сообщения
 LPCTSTR lpCaption,       //C-строка заголовка сообщения
 UINT uType               //Флаги окна сообщения
);
```

Значения флагов, управляющие набором кнопок, расположенных на окне:

- MB_ABORTRETRYIGNORE – Стоп, Отмена, Пропустить
- MB_OK – ОК,
- MB_OKCANCEL – ОК, Отмена,
- MB_RETRYCANCEL – Повтор, Отмена,
- MB_YESNO – Да, Нет,
- MB_YESNOCANCEL – Да, Нет, Отмена

Значения флагов, управляющие видом отображающейся иконки на окне сообщения:

- MB_ICONEXCLAMATION, MB_ICONWARNING – восклицательный знак;
- MB_ICONINFORMATION, MB_ICONASTERIX – символ i,
- MB_ICONQUESTION – знак вопроса,
- MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND – знак остановки.

Значения флагов, управляющие способом отображения окна сообщения:

- MB_APPLMODAL – окно hwnd переводится в неактивное состояние на время работы окна сообщения,
- MB_SYSTEMMODAL – на время работы окна сообщения все другие приложения в неактивном состоянии,
- MB_TASKMODAL – на время работы окна сообщения текущее приложение в неактивном состоянии. Если hwnd=NULL, то все перекрывающие окна,
- MB_HELP – добавляет кнопку «Справка»,

- MB_RIGHT – текст выравнивается по правому краю,
- MB_RTLREADING – отображает символы сообщения и текст заголовка в направлении справа налево,
- MB_SETFOREGROUND – окно сообщения выдвигается на передний план.

Ход работы

В рамках данной лабораторной работы необходимо, используя Win32API, разработать программу с графическим оконным интерфейсом согласно варианту задания. При разработке программы использовать мастера не рекомендуется. Выполнение лабораторной работы можно осуществлять в IDE Pelles C или Visual C++. Для разработки программы необходимо создать проект типа Win32 Application.

Варианты заданий:

1	После нажатия на левую клавишу мыши над рабочей областью окна в левом верхнем углу области отобразить временное окно размером в четверть области. Временное окно скрыть после отпускания клавиши в любом месте экрана.
2	В левом верхнем углу рабочей области окна создать дочернее окно. После нажатия левой клавиши мыши дочернее окно должно три раза «мигнуть». Для организации мигания использовать функцию FlashWindow.
3	Создать окно приложения размером в одну шестнадцатую площади экрана с заголовком «Форматирование диска» без кнопок изменения размеров, закрытия и сворачивания в пиктограмму. При перемещении курсора мыши над рабочей областью окно должно «убегать» от курсора мыши случайном образом выбранном направлении, не выходя за пределы экрана.
4	В левом верхнем углу рабочей области создать временное окно площадью в одну шестнадцатую этой области. При нажатии на левую клавишу мыши временное окно переместить в соседний по ходу часовой стрелки угол рабочей области.
5	При запуске i-го экземпляра (i>2) приложения спросить у пользователя, нужно ли его запустить. Если пользователь ответит «Да», то запустить его. Иначе на передний план переместить 2-й экземпляр приложения и завершить работу i-го экземпляра
6	Углы рабочей области окна приложения полностью занимают 4 временных окна одного класса. Если нажать левую клавишу мыши над временным окном, то это окно выдаст сообщение о своем заголовке.
7	В центре рабочей области окна располагается невидимое окно без заголовка размером в четверть площади рабочей области. После нажатия левой клавиши мыши над рабочей областью любого из окон окно без заголовка должно стать видимым, а после нажатия правой кнопки – невидимым.

8	В центре рабочей области окна отображено дочернее окно с вертикальными и горизонтальными полосами прокрутки размером в четверть этой области. Дочернее окно перемещается в тот угол рабочей области, где нажали левую клавишу мыши.
9	В центре рабочей области окна расположено окно без заголовка с вертикальной и горизонтальной полосами прокрутки размером в четверть рабочей области. При нажатии разных клавиш мыши временное окно выдает различные сообщения.
10	Создать окно размером в четверть площади экрана. После двойного щелчка мыши окно перемещается так, что его центр совпадает с координатами курсора мыши в момент щелчка.
11	Дочернее окно размером 100*100 пикселей при перемещении курсора мыши над ним «убегает» от курсора мыши в произвольном направлении, оставаясь в пределах рабочей области родительского окна.
12	При запуске второго экземпляра приложения сообщить о запрете запуска нескольких экземпляров, на передний план переместить первый экземпляр приложения, 3 раза изменив его подсветку (используя функцию FlashWindow), и завершить работу второго экземпляра.
13	В рабочей области окна приложения рядом друг с другом расположить 3 временных окна, каждое из которых по-своему реагирует на нажатие левой клавиши мыши.
14	Окно приложения занимает четверть экрана и расположено в левом верхнем углу. Создать временное окно такого же размера в правом нижнем углу. Любое окно после окна нажатия левой клавиши мыши перемещается в свободный по ходу часовой стрелки угол.
15	Центр рабочей области окна занимает временное окно размером в четверть площади области в свернутом состоянии. После нажатия левой клавиши мыши над рабочей областью временное окно распахивается в центре области, а после нажатия правой – сворачивается в центре.
16	Центр рабочей области окна занимает временное окно размером в четверть площади области. Оно перемещается в тот угол рабочей области, где щелкнули левой клавишей мыши. А после щелчка правой клавиши мыши временное окно перемещается в угол противоположный текущему углу.
17	Окно первого экземпляра приложения расположить в левом верхнем углу экрана, второго – в правом верхнем углу экрана, третьего – в левом нижнем углу, четвертого – в правом нижнем углу. Все окна одинаковых размеров и вместе занимают весь экран. В заголовке окна указать номер экземпляра. Запретить запуск пятого экземпляра.
18	Окно размером в четверть площади экрана расположено в центре экрана. После нажатия левой клавиши мыши окно несколько раз меняет подсветку и перемещается в угол экрана так, что курсор мыши оказывается за пределами экрана.

19	При запуске на первого экземпляра приложения выдать сообщение о количестве работающих копий этого приложения. Запустить экземпляр, только если согласен пользователь.
20	В рабочей области окна приложения рядом друг с другом расположить 4 временных окна, в заголовках которых указан номер окна. После нажатия левой клавиши мыши временное окно выдает сообщение, содержащее номер окна.
21	Правый верхний угол рабочей области окна приложения занимает временное окно размером в четверть этой области. После нажатия правой клавиши мыши над рабочей областью окна приложения временное окно сворачивается в пиктограмму в левом нижнем углу, а после нажатия левой– распахивается в правом верхнем углу рабочей области.
22	В углах рабочей области окна приложения созданы невидимые временные окна с заголовком. Каждое окно становится видимым после нажатия левой клавиши мыши над его частью рабочей области, и становится не видимым после нажатия правой клавиши мыши над его рабочей областью.
23	Окно приложения размером в четверть площади экрана занимает один из углов экрана. После нажатия левой клавиши мыши окно сворачивается в пиктограмму. После щелчка по пиктограмме оно восстанавливается в другом углу экрана.
24	При запуске приложения показать окна уде существующих копий этого приложения и спросить пользователя, нужно ли запустить еще один экземпляр. Если пользователь ответит «Да», то запустить его. Иначе завершить работу приложения.
25	После нажатия на правую клавишу мыши над рабочей областью окна в правом нижнем углу области отобразить временное окно размером в четверть области. Временное окно скрыть после отпускания клавиши в любом месте экрана.
26	В правом нижнем углу рабочей области окна создать дочернее окно. После нажатия правой клавиши мыши дочернее окно должно четыре раза «мигнуть». Для организации мигания использовать функцию FlashWindow.
27	В левом верхнем углу рабочей области создать временное окно площадью в одну шестнадцатую этой области. При нажатии на правую клавишу мыши временное окно переместить в соседний против хода часовой стрелки угол рабочей области.
28	В центре рабочей области окна отображено дочернее окно с вертикальными и горизонтальными полосами прокрутки размером в четверть этой области. Дочернее окно перемещается в угол рабочей области противоположный тому углу, в котором нажали правую клавишу мыши.

29	В центре рабочей области окна располагается невидимое окно без заголовка размером в четверть площади рабочей области. После нажатия правой клавиши мыши над рабочей областью любого из окон окно без заголовка должно стать видимым, а после нажатия левой кнопки– невидимым
30	Окно приложения занимает четверть экрана и расположено в правом верхнем углу. Создать временное окно такого же размера в левом нижнем углу. Любое окно после окна нажатия правой клавиши мыши перемещается в свободный против хода часовой стрелки угол.
31	После нажатия на левую правую мыши над рабочей областью окна в нижнем правом углу области отобразить временное окно размером в 1/6 часть области. Временное окно скрыть после отпускания клавиши в любом месте экрана.
32	В правом верхнем углу рабочей области окна создать дочернее окно. После каждого второго нажатия левой клавиши мыши на дочернее окно, окно должно три раза «мигнуть». Для организации мигания использовать функцию FlashWindow.
33	Создать окно приложения размером в одну десятую площади экрана с заголовком «Липучка» без кнопок изменения размеров, закрытия и сворачивания в пиктограмму. При перемещении курсора мыши над рабочей областью окно должно следовать за курсором с небольшим сдвигом влево.
34	В левом верхнем углу рабочей области создать временное окно площадью в одну десятую этой области. При нажатии на правую клавишу мыши временное окно переместить в соседний против часовой стрелки угол рабочей области.
35	При запуске i-го экземпляра (i>2) приложения спросить у пользователя, нужно ли его запустить. Если пользователь ответит «Да», то запустить его. Иначе на передний план переместить 1-й экземпляр приложения и «мигнуть» 1-й экземпляр 2 раза при помощи функции FlashWindow.
36	Углы рабочей области окна приложения полностью занимают 4 временных окна одного класса. Если нажать левую клавишу мыши над временным окном, то это окно выдаст сообщение о своем размере.
37	При запуске не первого экземпляра приложения выдать сообщение о количестве работающих копий этого приложения. Завершить текущую копию приложения, уведомить об этом пользователя и запустить новый экземпляр приложения.
38	В центре рабочей области окна отображено дочернее окно с вертикальными и горизонтальными полосами прокрутки размером в четверть этой области. Дочернее окно перемещается в тот угол рабочей области, где нажали левую клавишу мыши, выдавая при этом сообщение вида: «Переместилось с верхнего левого угла, в нижний правый».

39	В центре рабочей области окна расположено окно без заголовка в четверть рабочей области. При нажатии разных клавиш мыши временное окно выдает различные сообщения и перемещается по направлению часовой стрелки по углам рабочей области.
40	Создать окно размером в шестнадцатую часть площади экрана. После щелчка мыши окно перемещается так, что его центр совпадает с координатами курсора мыши в момент щелчка.

Контрольные вопросы

1. Для чего предназначена библиотека Win32 API.
2. Объясните концепцию событийного программирования.
3. Что такое сообщение и какими параметрами оно задается?
4. Перечислите параметры функции WinMain.
5. Опишите основной алгоритм выполнения функции WinMain.
6. Перечислите поля структуры WNDCLASS.
7. Какая функция осуществляет создание окна приложения?
8. Какие функции используются для отображения окна приложения?
9. Какие функции используются для получения и обработки сообщений?
10. Для чего нужна функция обработчик и какому прототипу она должна удовлетворять?

Защита лабораторной работы

Для защиты лабораторной работы студент предоставляет рабочую программу и исходный код. Показывает и рассказывает преподавателю о выполненном задании.

Если все требования, указанные в пункте «Ход работы» выполнены в полной мере, получены ответы на вопросы преподавателя и, при необходимости, сделано дополнительное задание, то студент получает 60 баллов.

За невыполнение задания в полной мере преподаватель отнимает баллы:

- 10 баллов за каждую ошибку в программе;
- 10 баллов за невыполнение одного из требований лабораторной работы;
- 5 баллов за неполный ответ на вопрос;
- 3 балла за каждый недочет в работе программы.

В случае, если студент применил творческий подход к выполнению работы, преподаватель может поставить до 20 бонусных баллов.

Лабораторная работа считается не защищенной в случаях, если студент:

- не разбирается в коде программы;
- не может ответить на вопросы преподавателя по работе программы;
- не выполняет дополнительное задание (схожее с уже написанным);
- пытается сдать чужой вариант.

В таких случаях преподаватель выдает *новый вариант* и отнимает до *30 баллов* рейтинга.