Лабораторная работа №2

студента группы ИТ – 32

Курбатовой Софьи Андреевны

Выполнение:_____ Защита _____

ОКОННЫЕ ПРИЛОЖЕНИЯ WINDOWS

Цель работы: получение практических навыков создания оконных приложений Windows на языке C/C++ с применением Win32 API.

Содержание работы

Вариант 4

1. Изучить элементы управления общего пользования, указанные в варианте задания. Особое внимание уделить тому, какие стили они поддерживают, какие сообщения принимают и какие уведомления отправляют родительскому окну. Включить в отчет название и описание изученных элементов управления.

2. Изучить диалоговые окна общего пользователя, указанные в варианте задания. Включить в отчет назначение и описание изученных диалоговых окон.

3. Изучить оконное сообщение WM_TIMER в сообщения, указанные в варианте задания. Включить в отчет описание изученных оконных сообщений.

4. Разработать в Visual C++ оконное приложение Win32, которое:

– должно создавать главное окно, содержащее меню и элементы управления, указанные в варианте задания;

– должно обрабатывать комбинации быстрых клавиш;

– должно создавать одно или несколько диалоговых окон с использованием шаблона;

– должно создавать диалоговое окно сообщений;

– должно создавать диалоговые окна, указанные в варианте задания;

– должно обрабатывать оконное сообщение WM_TIMER и оконные сообщения, указанные в варианте задания.

5. Протестировать работу приложения, разработанного в п.4 на компьютере под управлением Windows. Результаты тестирования отразить в отчете.

6. Включить в отчет исходный программный код и выводы о проделанной работе.

4,19	Календарь (Month calendar) Редактируемое поле (Edit box) Список (List box) Счетчик или стрелки (Spin)	Сохранить как (Save As) Заменить (Replace)	WM_LBUTTONDBLCLK WM_LBUTTONDOWN WM_LBUTTONUP WM_MOUSEWHEEL WM_SIZING WM_SYSCHAR WM_SYSKEYUP

Рис. 2.1. задание для варианта 4

Ход работы

1. Разница между элементами управления общего пользования и предопределенными элементами управления (кнопка) состоит в том, какие сообщения они посылают для уведомления. Предопределенные элементы управления посылают уведомляющие сообщения WM_COMMAND, в то время как элементы управления общего пользования посылают сообщения WM_NOTIFY.

Первые сообщения называемые уведомительными сообщениями, информируют процедуру, что пользователь ввел данные и дал разрешение на выполнение присущей ей работы. Вторые отправляется общим элементом управления в его родительское окно, когда произошло событие или элементу управления требуется некоторая информация.

Различные описания, которые необходимы для использования элементов управления общего пользования, определяются в файле commctrl.h и в библиотеке ComCTL32.dll.

Каждый элемент управления - это дочернее окно, и поэтому должно иметь стиль WS_CHILD. Чтобы гарантировать, что элемент управления видимый, когда отображается диалоговое окно, каждый элемент управления должен иметь также стиль WS_VISIBLE. Другие, обычно используемые стили окна, - это WS_BORDER для элементов управления, которые не обязательно имеют рамки, WS_DISABLED для элементов управления, которые должны быть отключены, когда создается первоначальное диалоговое окно и WS_TABSTOP и WS_GROUP для элементов управления, к которым можно обращаться, используя клавиатуру.

Элементы управления, рассматриваемые в лабораторной работе: Календарь (Month calendar), Редактируемое поле (Edit box), Список (List Box), Счетчики или стрелки (Spin).

Календарь – оконный класс SySMonthCall2, префикс флага стиля MCS. Тип отправляемых сообщений WM_NOTIFY. Элемент управления демонстрирует пользователю календарь. Среди поддерживаемых **стилей** можно выделить:

MCS_DAYSTATE - отправляет уведомления MCN_GETDAYSTATE, чтобы запросить информацию о том, какие дни должны быть выделены.

MCS_MULTISELECT – позволяет пользователю определять диапазон видимой даты. Можно изменить максимальный диапазон, который может быть выбран с помощью сообщения MCM_SETMAXSELCOUNT. Так, например, на рисунке 2.2 выделен диапазон – неделя, а день 29 сентября.

MCS_WEEKNUMBERS – календарь отображает номера недель (1-52) слева от каждой строки дней. Неделя 1 определяется как первая неделя, содержащая не менее четырех дней.

MCS_NOTODAYCIRCLE – календарь не обводит дату «сегодня».

MCS_NOTODAY - элемент управления не отображает дату «сегодня» в нижней части элемента управления.

Среди отправляемых **сообщений** можно выделить следующие:

MCM_GETCALENDARBORDER - возвращает размер границы в пикселях. Можно отправить это сообщение явно или с помощью макроса MonthCal_GetCurrentView. Параметры: wParam должен быть 0, lParam должен быть 0. Возвращаемое значение: размер границы в пикселях.

MCM_GETFIRSTDAYOFWEEK - извлекает первый день недели для элемента управления. Можно отправить это сообщение явно или с помощью макроса MonthCal_GetFirstDayOfWeek. . Параметры: wParam должен быть 0, lParam должен быть 0. Возвращаемое значение имеет тип DWORD. Первая часть «слова» содержит 0, если первый день недели соответствует региональным установкам (LOCALE_IFFIRSTDAYOFWEEK) и 1 в ином случае. Вторая часть «слова» представляет собой значение типа INT: 0 – понедельник, 1- вторник и т.д.

MCM_GETMONTHRANGE - извлекает информацию о дате (используя структуры SYSTEMTIME), которая представляет собой верхний и нижний пределы отображения элемента управления. Можно отправить это сообщение явно или с помощью макроса MonthCal_GetMonthRange. Параметры: wParam здесь значение, указывающее область действия извлекаемых пределов диапазона. Это значение должно быть одним из следующих: GMR_DAYSTATE – включаем предыдущие и конечные месяцы видимого диапазона, которые отображаются только частично, GMR_VISIBLE – включаем только те месяцы, которые полностью отображаются. lParam – указатель на двухэлементный массив структур SYSTEMTIME, который будет получать нижнюю и верхнюю границы области, заданной wParam. Нижний и верхний пределы помещаются в lprgSysTimeArray[0] и lprgSysTimeArray[1] соответственно. Этот параметр должен быть действительным адресом и не может быть нулевым. Возвращаемое значение, имеет тип INT, представляет диапазон в месяцах, охватываемый двумя ограничениями, возвращаемыми в lParam.

MCM_SETTODAY – позволяет выбрать, что установить в «сегодня». Можно отправить это сообщение явно или с помощью макроса MonthCal_SetToday. Параметры: wParam должен быть 0, lParam должен быть указателем на структуру SYSTEMTIME, содержащую дату, которая должна быть установлена. В ином случае будет установлено значение по умолчанию. Возвращаемое значение: не используется.

MCM_SETCURSEL - устанавливает текущую выбранную дату для элемента. Если указанная дата не отображается, элемент управления обновляет дисплей, чтобы отобразить ее. Можно отправить это сообщение явно или с помощью макроса MonthCal_SetCurSel. Параметры: wParam должен быть 0, lParam должен быть указателем на SYSTEMTIME, которая содержит дату, которая должна быть выделена. Возвращаемое значение: не 0 в случае успеха и 0 в ином случае. Это сообщение завершится ошибкой, если оно будет применено к элементу управления календарем месяца, использующему MCS_MULTISELECTstyle.

Среди отправляемых **уведомлений**:

MCN_GETDAYSTATE - Отправляется элементом управления календарем месяца для запроса информации о том, как должны отображаться отдельные дни. Этот код уведомления отправляется только элементами управления календарем месяцев, использующими стиль MCS_DAYSTATE, и отправляется в виде сообщения WM_NOTIFY. Параметр: lparam - указатель на структуру NMDAYSTATE. Структура содержит информацию о временном интервале, для которого элемент управления нуждается в информации, и получает адрес массива, который предоставляет эти данные. Возвращаемое значение: нет.

MCN_SELCHANGE - отправляется элементом управления календарем месяца при изменении текущей выбранной даты или диапазона дат. Этот код уведомления отправляется в виде сообщения WM_NOTIFY. Параметр: lParam - указатель на

структуру NMSELCHANGE, содержащую информацию о текущем выбранном диапазоне дат. Возвращаемое значение: нет.

MCN_SELECT – отправляется элементом управления календарем месяца, когда пользователь делает явный выбор даты в элементе управления календарем месяца. Этот код уведомления отправляется в виде сообщения WM_NOTIFY. Параметр: lParam - указатель на структуру NMSELCHANGE, содержащую информацию о текущем выбранном диапазоне дат. Возвращаемое значение: нет.

MCN_VIEWCHANGE – отправляется элементом управления при изменении текущего представления. Этот код уведомления отправляется в виде сообщения WM_NOTIFY. Параметр: lParam указывает на структуру ONVIEWCHANGE, содержащую информацию о текущем представлении. Возвращаемое значение: нет.

NM_RELEASEDCAPTURE (monthcal) - уведомляет родительское окно элемента управления месячным вызовом о том, что элемент управления освобождает захват мыши. Этот код уведомления отправляется в виде сообщения WM_NOTIFY. Параметр: lparam указывает на структуру NMHDR, содержащую дополнительную информацию об этом уведомлении. Возвращаемое значение: игнорируется.

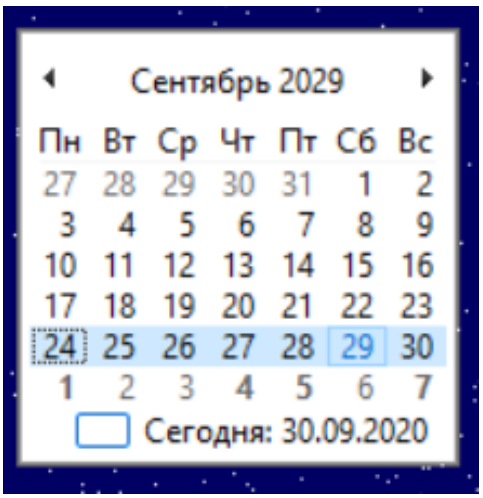

Рис. 2.2. Календарь

Редактируемое поле – оконный класс Edit, префикс флага стиля – ES. Тип отправляемых сообщений WM_Command. Это прямоугольное окно управления, обычно используемое в диалоговом окне для ввода и редактирования текста пользователем. Продемонстрирован на рисунке 2.4 . Среди поддерживаемых **стилей** можно выделить такие как:

ES_MULTILINE – позволяет переносить вводимый в элемент управления текст на новую строку.

ES_NUMBER – разрешен ввод только цифр.

ES_CENTER – выравнивание текста по центру элемента управления.

ES_LEFT – выравнивание по левому краю.

ES_READONLY - Запрещает пользователю вводить или редактировать текст в элементе управления редактированием. Чтобы изменить этот стиль после создания элемента управления, используется сообщение EM_SETREADONLY.

Среди отправляемых **сообщений** можно выделить следующие:

EM_GETCUEBANNER – возвращает текст , отображаемый в качестве текстовой подсказки или подсказки в элементе управления редактированием.

Параметр: wParam – указатель на буфер, который получает набор текста в качестве текстового сигнала. Вызывающий объект отвечает за выделение буфера., lParam – размер буфера, на который указывает параметр wParam в элементах wchar, включая завершающий нуль. Возвращаемое значение: TRUE или FALSE.

EM_SETREADONLY - устанавливает или удаляет стиль только для чтения (ES_READONLY). Это сообщение можно отправить либо в элемент управления редактированием, либо в расширенный элемент управления редактированием. Параметр: wParam - указывает, следует ли установить или удалить стиль ES_READONLY, значение TRUE или FALSE. Возвращаемое значение: ненулевое, если установка стиля прошла успешно.

EM_GETMODIFY – возвращает состояние флага модификации элемента управления редактированием. Флаг указывает, было ли изменено содержимое элемента управления Edit. Это сообщение можно отправить либо в элемент управления редактированием, либо в расширенный элемент управления редактированием. Параметры: wParam должен быть 0, lParam должен быть 0. Возвращаемое значение: ненулевое, если элемент был изменен.

EM_SCROLL – прокрутка текста по вертикали в многострочном элементе управления редактированием. Это сообщение эквивалентно отправке сообщения WM_VSCROLL в элемент управления edit. Это сообщение можно отправить либо в элемент управления редактированием, либо в расширенный элемент управления редактированием. Параметры: wParam – описывает действие, которое полоса прокрутки должна выполнить и значения могут быть следующими: SB_LINEDOWN/SB_LINEUP– прокрутить на одну строку вниз/вверх, SB_PAGEDOWN/SB_PAGEUP – прокрутить на одну страницу вниз/вверх. Возвращаемое значение: если сообщение успешно, то слово возвращаемого значения равно TRUE, а LOWORD-это количество строк, которые прокручивает команда. Возвращаемое число может отличаться от фактического количества прокрученных строк, если прокрутка перемещается в начало или конец текста. Если параметр wParam указывает недопустимое значение, то возвращаемое значение равно FALSE.

EM_SETLIMITTEXT – задает ограничение текст, т.е. устанавливает максимальное количество текста, которое пользователь может ввести в элемент управления. Это сообщение можно отправить либо в элемент управления редактированием, либо в расширенный элемент управления редактированием. Параметр: wParam – в него записывается какое максимальное количество символов пользователь может ввести. Для ANSI это количество байт, для UNICODE это количество элементов. Это число не содержит нуль-символа. Возвращаемое значение: нет.

Поддерживаются такие **уведомления** как:

EN_CHANGE – отправляется, когда пользователь предпринял действие, которое, возможно, изменило текст в элементе управления редактированием. В отличие от кода уведомления EN_UPDATE, этот код уведомления отправляется после обновления экрана системой. Родительское окно элемента управления Edit получает этот код уведомления через сообщение WM_COMMAND. Параметр wParam - LOWORD содержит идентификатор элемента управления edit. Это слово указывает код уведомления, lParam – дескриптор элемента управления редактированием.

EN_UPDATE – отправляется, когда элемент управления редактированием собирается перерисовать себя. Этот код уведомления отправляется после того, как элемент управления отформатировал текст, но до его отображения. Это позволяет при необходимости изменить размер окна управления редактированием. Родительское окно элемента управления edit получает этот код уведомления через сообщение WM_COMMAND. Параметр wParam - LOWORD содержит идентификатор элемента управления Edit. Это слово указывает код уведомления, lParam – дескриптор элемента управления редактированием.

EN_ERRSPACE - отправляется, когда элемент управления редактированием не может выделить достаточно памяти для удовлетворения конкретного запроса. Родительское окно элемента управления Edit получает этот код уведомления через сообщение WM_COMMAND. Параметр wParam - LOWORD содержит идентификатор элемента управления Edit. Это слово указывает код уведомления, lParam – дескриптор элемента управления редактированием.

EN_HSCROLL – отправляется, когда пользователь щелкает горизонтальную полосу прокрутки элемента управления редактирования. Родительское окно элемента управления Edit получает этот код уведомления через сообщение WM_COMMAND. родительское окно получает уведомление до обновления экрана. Параметр wParam - LOWORD содержит идентификатор элемента управления Edit. Это слово указывает код уведомления, lParam – дескриптор элемента управления редактированием.

EN_KILLFOCUS - отправляется, когда элемент управления теряет фокус клавиатуры. Родительское окно элемента управления Edit получает этот код уведомления через сообщение WM_COMMAND. Параметр wParam - LOWORD содержит идентификатор элемента управления Edit. Это слово указывает код уведомления, lParam – дескриптор элемента управления редактированием.

EN_SETFOCUS - отправляется, когда элемент управления редактированием получает фокус клавиатуры. Родительское окно элемента управления Edit получает этот код уведомления через сообщение WM_COMMAND. Параметр wParam - LOWORD содержит идентификатор элемента управления Edit. Это слово указывает код уведомления, lParam – дескриптор элемента управления редактированием.

EN_MAXTEXT – отправляется, когда текущая вставка текста превысила заданное количество символов для элемента управления редактированием. Вставка текста при этом была усечена. Этот код уведомления также отправляется, когда элемент управления редактированием не имеет стиля ES_AUTOHSCROLL и количество вставляемых символов превышает ширину элемента управления редактированием. Этот код уведомления также отправляется, когда элемент управления редактированием не имеет стиля ES_AUTOVSCROLL и общее количество строк, полученных в результате вставки текста, превышает высоту элемента управления редактированием.Родительское окно элемента управления Edit получает этот код уведомления через сообщение WM_COMMAND. Параметр wParam - LOWORD содержит идентификатор элемента управления Edit. Это слово указывает код уведомления, lParam – дескриптор элемента управления редактированием.

Список – оконный класс ListBox, префикс флага стиля – LBS_. Тип отправляемых сообщений WM_Command. Этот элемент содержит простой список, из которого пользователь обычно может выбрать один или несколько элементов. Коды

уведомлений списка отправляются, когда пользователь выбирает, дважды щелкает (LBN_DBLCLK) или отменяет элемент списка (LBN_SELCANCEL); когда список получает или теряет фокус клавиатуры (LBN_KILLFOCUS); и когда система не может выделить достаточно памяти для запроса списка (LBN_ERRSPACE). Сообщение WM_COMMAND содержит идентификатор списка в младшем слове параметра wParam и код уведомления в старшем слове. Параметр lParam содержит дескриптор окна управления. Продемонстрирован на рисунке 2.4. Среди поддерживаемых **стилей** можно выделить такие как:

LBS_WANTKEYBOARDINPUT – указывает , что владелец списка получает сообщения WM_VKEYTOITEM всякий раз, когда пользователь нажимает клавишу и список оказывается в фокусе. Это позволяет приложению выполнять обрабатывать ввод с клавиатуры.

LBS_SORT – сортировка будет осуществлена в алфавитном порядке.

LBS_NOTIFY – вызывает отправку кода уведомления в родительское окно всякий раз, когда пользователь щелкает элемент списка (LBN_SELCHANGE), дважды щелкает элемент (LBN_DBLCLK) или отменяет выбор (LBN_SELCANCEL).

LBS_MULTIPLESEL – включает или выключает выбор строки каждый раз, когда пользователь щелкает или дважды щелкает строку в списке. Пользователь может выбрать любое количество строк.

LBS_COMBOBOX – уведомляет list box о том, что он является частью поля со списком. Это позволяет координировать работу двух элементов управления, чтобы они представляли собой единый пользовательский интерфейс. Этот стиль должен быть задан самим полем со списком. Если стиль задается чем-либо, кроме поля со списком, поле со списком будет неправильно считать себя дочерним элементом поля со списком, и это приведет к сбою.

Элементу управления можно отправлять такие **сообщения** как:

LB_ADDSTRING – добавляет новую строку. Если элемент управления без стиля LBS_SORT, то строка будет добавлена в конец списка. Параметр: wParam не используется, lParam – является указателем на строку с нулевым завершением, которая должна быть добавлена. Если в списке есть стиль, нарисованный владельцем, но не стиль строк LBS_HAS, этот параметр сохраняется в виде данных элемента, а не строки. Возвращаемое значение: нулевой индекс строки в списке. Если возникает ошибка, то возвращаемое значение равно LB_ERR. Если есть недостаточное пространство, чтобы сохранить новую строку, то возвращается значение LB_ERR пространства.

LB_DELETESTRING – удаляет строку. Параметр: wParam – нулевой индекс удаленной строки, lParam – не используется. Возвращаемое значение: количество оставшихся в списке элементов. LB_ERR, если параметр wParam указывает индекс, превышающий количество элементов в списке.

LB_FINDSTRING – возвращает индекс первой строки в списке, которая начинается с указанной строки. Параметр: wParam – нулевой индекс элемента перед первым элементом, подлежащим поиску. Когда поиск достигает нижней части списка, он продолжает поиск из верхней части списка обратно к элементу, указанному параметром wParam. Если значение wParam равно -1, то поиск выполняется по всему списку с самого начала, lParam – указатель на null-завершающую строку, содержащую строку, для которой выполняется поиск. Поиск

не зависит от регистра, поэтому эта строка может содержать любую комбинацию прописных и строчных букв. Возвращаемое значение: индекс, совпавшей строки или LB_ERR.

LB_FINDSTRINGEXACT – находит первую строку списка, которая точно соответствует указанной строке, за исключением того, что поиск не чувствителен к регистру. Параметр: wParam – нулевой индекс элемента перед первым элементом, подлежащим поиску. Когда поиск достигает нижней части списка, он продолжает поиск из верхней части списка обратно к элементу, указанному параметром wParam. Если значение wParam равно -1, то поиск выполняется по всему списку с самого начала. lParam – указатель на null-завершающую строку, содержащую строку, для которой выполняется поиск. Поиск не зависит от регистра, поэтому эта строка может содержать любую комбинацию прописных и строчных букв. Возвращаемое значение: индекс, совпавшей строки или LB_ERR.

LB_GETCURSEL – возвращает индекс текущего выбранного элемента, если таковые имеются, то в списке с одним выбором. Параметры: wParam должен быть 0, lParam должен быть 0. Возвращаемое значение: в списке с одним выбором возвращаемым значением является нулевой индекс текущего выбранного элемента. Если выбор отсутствует, то возвращаемое значение равно LB_ERR.

LB_SETITEMDATA – связывает значение с элементом списка. Параметр: wParam указывает нулевой индекс элемента. Если это значение равно -1, то значение lParam применяется ко всем элементам списка, lParam – указывает значение, которое будет связано с элементом. Возвращаемое значение: в случае ошибки LB_ERR.

Поддерживаются такие **уведомления** как:

WM_DELETEITEM – отправляется владельцу списка или поля со списком при уничтожении списка или поля со списком или при удалении элементов с помощью сообщения LB_DELETESTRING, LB_RESETCONTENT, CB_DELETESTRING или CB_RESETCONTENT. Система отправляет сообщение WM_DELETEITEM для каждого удаленного элемента. Система отправляет сообщение WM_DELETEITEM для любого удаленного элемента списка или combobox с ненулевыми данными элемента. Параметр: wParam задает идентификатор элемента управления, отправившего сообщение WM_DELETEITEM, lparam – является указателем на структуру DELETEITEMSTRUCT, содержащую информацию об элементе, удаленном из списка. Возвращаемое значение: приложение вернет TRUE после обработки.

WM_VKEYTOITEM – отправляется списком со стилем ввода клавиатуры LBS_WANT своему владельцу в ответ на сообщение WM_KEYDOWN. Параметр: wParam состоит из двух частей: LOWORD указывает код виртуальной клавиши, которую нажал пользователь. HIWORD указывает текущее положение курсора, Параметр lParam содержит дескриптор списка. Возвращаемое значение указывает действие, которое приложение выполнило в ответ на сообщение. Возвращаемое значение: 2 указывает на то, что приложение обработало все аспекты выбора элемента и не требует дальнейших действий со стороны списка. 1 указывает, что поле списка должно выполнить действие по умолчанию в ответ на нажатие клавиши. 0 или больше указывает индекс элемента в списке и указывает, что список должен выполнить действие по умолчанию для нажатия клавиши на указанном элементе.

LBN_SELCHANGE – уведомляет приложение о том, что выбор в списке изменился в результате ввода пользователем данных. Родительское окно списка получает этот код уведомления через сообщение WM_COMMAND. Параметр: wParam состоит из двух частей LOWORD содержит идентификатор list box, HIWORD определяет код отправляемого уведомления, lParam – содержит дескриптор списка.

LBN_DBLCLK – уведомляет приложение о том, что пользователь дважды щелкнул элемент в списке. Родительское окно списка получает этот код уведомления через сообщение WM_COMMAND. Параметр: wParam состоит из двух частей LOWORD содержит идентификатор list box, HIWORD определяет код отправляемого уведомления, lParam – содержит дескриптор списка.

LBN_SELCANCEL – уведомляет приложение о том, что пользователь отменил выбор в списке. Родительское окно списка получает этот код уведомления через сообщение WM_COMMAND. Параметр: wParam состоит из двух частей LOWORD содержит идентификатор list box, HIWORD определяет код отправляемого уведомления, lParam – содержит дескриптор списка.

Счетчики или стрелки Spin – msctls_updown32, префикс флага стиля – UDS. Тип отправляемых сообщений WM_NOTIFY. Элемент управления «вверх-вниз» - это пара кнопок со стрелками, которые пользователь может нажимать для увеличения или уменьшения значения, например положения прокрутки или числа, отображаемого в сопутствующем элементе управления (см. рисунок 2.5).

По умолчанию может управлять, например такими **сообщениями** как:

WM_CREATE – выделяет и инициализирует частную структуру данных и сохраняет ее адрес в виде данных окна.

WM_DESTROY – освобождает данные, выделенные во время обработки WM_CREATE.

WM_TIMER – изменяет текущее положение, если мышь удерживается над кнопкой и прошел достаточный интервал времени.

Отправляемые **сообщения**:

UDM_GETACCEL – извлекает информацию об ускорении (здесь: быстрые клавиши) для управления вверх-вниз. Параметр wParam - количество элементов в массиве, заданном параметром lParam, lParam указатель на массив структур UDACCEL, получающих информацию об ускорении. Возвращаемое значение: количество ускорителей, установленных в данный момент для элемента управления.

UDM_GETBASE – возвращает текущее основание системы счисления (то есть, либо с основанием 10 или 16) на стрелки вверх-вниз. Параметры: wParam должен быть 0, lParam должен быть 0. Возвращаемое значение: основание системы счисления.

UDM_GETBUDDY – возвращает дескриптор в текущее окно «приятеля». Параметры: wParam должен быть 0, lParam должен быть 0. Возвращаемое значение: дескриптор окна «приятеля».

UDM_GETPOS – извлекает текущее положение элемента управления вверх-вниз с 16-битной точностью. Параметры: wParam должен быть 0, lParam должен быть 0. Возвращаемое значение: в случае успеха HIWORD будет установлен в 0, а LOWORD получит значение текущей позиции. В случае ошибки HIWORD установится в ненулевое значение.

UDM_SETACCEL – устанавливает «ускорители» для элемента управления. Параметр: wParam – количество UDACCEL структур, lParam указатель на массив UDACCEL структур, который содержит информацию об «ускорителях». Элементы должны быть отсортированы в порядке возрастания на основе члена nSec. Возвращаемое значение: True или False.

Среди поддерживаемых **стилей** можно выделить такие как:

*UDS_ARROWKEY*S – заставляет элемент управления up-down увеличивать и уменьшать положение при нажатии клавиш со стрелками вверх и вниз.

UDS_SETBUDDYINT – заставляет элемент управления устанавливать текст родственного окна (с помощью сообщения WM_SETTEXT) при изменении позиции. Текст состоит из позиции, отформатированной в виде десятичной или шестнадцатеричной строки.

UDS_AUTOBUDDY – автоматический выбор родственного окна.

UDS_WRAP – вызывает «обертывание» позиции, если она увеличивается или уменьшается за пределами конца или начала диапазона.

UDS_HORZ – заставляет стрелки управления вверх-вниз указывать влево и вправо, а не вверх и вниз.

Среди отправляемых **уведомлений** можно выделить такие как:

NM_RELEASEDCAPTURE (up-down) – уведомляет родительское окно элемента управления up-down о том, что элемент управления освобождает захват мыши. Этот код уведомления отправляется в виде сообщения WM_NOTIFY. Параметр: lParam - указатель на структуру NMHDR, содержащую дополнительную информацию об этом уведомлении. Возвращаемое значение: игнорируется.

UDN_DELTAPOS – отправляется операционной системой в родительское окно элемента управления вверх-вниз, когда положение элемента управления вот-вот изменится. Это происходит, когда пользователь запрашивает изменение значения, нажав на стрелку вверх или вниз элемента управления. Этот код уведомления отправляется в виде сообщения WM_NOTIFY. Параметр: lParam – указатель на структуру NMUPDOWN, содержащую информацию об изменении позиции. Член iPos этой структуры содержит текущее положение элемента управления. Дельта-член структуры - это целое число со знаком, содержащее предлагаемое изменение позиции. Если пользователь нажал кнопку вверх, это положительное значение. Если пользователь нажал кнопку вниз, это отрицательное значение. Возвращаемое значение: ненулевое значение, чтобы предотвратить изменение положения элемента управления, или ноль, чтобы разрешить это изменение.

Чтобы использовать оконные классы для этих элементов следует вызвать функцию InitCommonControlsEx, которая регистрирует оконные классы этих элементов управления.

2. Различают модальные и немодальные диалоговые окна. Первые не позволяют работать с другими окнами приложения, пока работа с модальным окном не будет завершена. Вторые позволяют свободно переключаться между окнами одного приложения.

К диалоговым окнам общего пользования можно отнести окна «Открыть файл», «Сохранить Как», «Найти и Заменить», «Печать» и т.д. Специальные функции для их открытия определен в COMDLG32.dLL и имеют следующее объявление: GetOpenFileName, GetSaveFileName, FindText, ReplaceText, PrintDlg, ChooseColor,

ChooseFont. Для использования этих функций в первую очередь необходимо проинициализировать поля соответствующей структуры и передать функции указатели на эту структуру. Функции создают и отображают окно диалога. Когда пользователь закрывает окно диалога, то в соответствующем поле структуры будет содержаться его выбор.

Так, для открытия **окна «Заменить»** (рисунок 2.7)используется структура FINDREPLACE. Она содержит информацию, которую используют функции FindText и ReplaceText, чтобы инициализировать диалоговые окна Найти (Find) и Заменить (Replace). Зарегистрированное сообщение FINDMSGSTRING использует эту структуру, чтобы передать введенные данные пользовательского поиска или замены в окно владельца блока диалога. Структура содержит такие поля как:

DWORD lStructSize - определение длины структуры в байтах.

HWND hwndOwner – дескриптор окна, которое владеет диалоговым окном. Оконная процедура заданного окна принимает сообщения FINDMSGSTRING от блока диалога. Не может быть NULL.

DWORD Flags – используются для инициализации диалогового окна и устанавливает, когда диалог отправляет сообщение, чтобы обозначить ввод данных пользователем. Например, если в сообщении установлен FR_FINDNEXT, то это означает, что пользователь нажал на кнопку «Найти далее». Тогда, lpstrFindWhat определит строку для поиска. Если установлен FR_REPLACE, то это означает, что пользователь щелкнул по кнопке Заменить (Replace) в диалоговом окне Заменить (Replace). Член lpstrFindWhat определяет строку, которая заменяется, а член lpstrReplaceWith определяет строку замены.

HINSTANCE hInstance – дескриптор объекта памяти, содержащего шаблон окна или дескриптор модуля, содержащего шаблон диалогового окна. Зависит от выбранного флага: FR_ENABLETEMPLATEHANDLE или FR_ENABLETEMPLATE соответственно. Может быть NULL.

LPTSTR lpstrFindWhat - указатель на буфер, который использует сообщение FINDMSGSTRING для передачи строки поиска с нуль-терминатором в конце, которую пользователь набрал в редактируемом поле «Что искать ».

LPTSTR lpstrReplaceWith Указатель на буфер, который использует сообщение FINDMSGSTRING для передачи строки поиска с нуль-терминатором в конце, которую пользователь набрал в редактируемом поле «Заменить на».

WORD wFindWhatLen, WORD wReplaceWithLen – длина буфера в байтах для указанных выше строк.

LPARAM lCustData – устанавливает определяемые программой данные, которые система передает в фильтр (hook) - процедуру, идентифицированную при помощи члена lpfnHook.

LPFRHOOKPROC lpfnHook – указатель на фильтр (hook) - процедуру FRHookProc, которая может обрабатывать сообщения, предназначенные для диалогового окна. Этот член игнорируется, если в члене Flags не установлен флажок FR_ENABLEHOOK.

LPCTSTR lpTemplateName – указатель на строку с символом нуля в конце, которая именует ресурс шаблона диалогового окна в модуле, идентифицированном членом hInstance. Этот шаблон заменяет стандартный шаблон диалогового окна.

Для открытия **окна «Сохранить как»** (рисунок 2.9) используется структура OPENFILENAME. В структуре выделим такие важные поля как: \

HWND hwndOwner - идентифицирует окно, которое является «владельцем». Это может быть как указатель на окно, так и NULL значение (если нет владельца).

LPCTSTR lpstrFilter - указатель на буфер, в котором находятся пары нуль-терминированных (null-terminated) строк для фильтра. Последняя строка в этом буфере должна заканчиваться двумя NULL символами. Первая строка пары содержит описание фильтра (например, "Text Files"), а вторая - сам шаблон фильтра (например, "*.TXT"). Можно указать несколько шаблонов для одного фильтра. Например: *"Bitmap files(*.bmp)\0*.bmp\0JPEG files(*.jpg)\0*.jpg\0All files(*.*)\0*.*\0\0"*

LPCTSTR lpstrTitle - указатель на строку, которая будет помещена в заголовке окна диалога. Если этот параметр имеет значение NULL, то система использует стандартные заголовки (соответственно, «Сохранить как» или «Открыть»).

Так же следует выделить и ряд полезных **флагов**, используемых при инициализации диалога:

OFN_CREATEPROMPT - если пользователь указал файл, которого не существует, то диалог запросит разрешение создать новый файл с указанным именем. Если пользователь выберет создание нового файл, то диалог закроется и вернет указанное имя, в противном случае диалог останется открытым.

OFN_OVERWRITEPROMPT - определяет, что диалог "Сохранить как" выдаст предупреждающее сообщение, если файл уже существует. Пользователь должен подтвердить перезапись файла.

OFN_EXPLORER - диалог будет использовать Explorer-style интерфейс диалога.

OFN_EXTENSIONDIFFERENT – пользователь ввел расширение имени файла, которое отличается от расширения, определенного в поле lpstrDefExt.

3. Уведомительное сообщение **WM_TIMER** помещается в очереди сообщений установленного потока, когда у таймера истекает время. Сообщение помещается функцией GetMessage или PeekMessage. Приложение может заказать для любого своего окна несколько таймеров, которые будут периодически посылать в функцию окна сообщение с кодом WM_TIMER. Параметр wParam сообщения WM_TIMER содержит идентификатор таймера, который был указан или получен от функции SetTimer при создании таймера. С помощью параметра lParam можно определить адрес функции, которая обрабатывает сообщения таймера. Сообщение WM_TIMER - сообщение с низким приоритетом. Функции GetMessage и PeekMessage помещают это сообщение в очередь только тогда, когда, никакие другие высокоприоритетные сообщения не находятся в очереди сообщений потока.

Уведомительное сообщение **WM_SIZING** отправляется окну, у которого пользователь изменяет размеры. Обрабатывая это сообщение, прикладная программа может контролировать размер и позицию перетаскиваемого прямоугольника и, если необходимо, изменять его размер или позицию. Параметр: wParam определяет край окна, который измеряется. Этот параметр может иметь одно из следующих значений: WMSZ_LEFT/WMSZ_RIGHT – левая/правая граница, WMSZ_TOP/ WMSZ_BOTTOM – верхняя/нижняя граница, WMSZ_TOPLEFT/WMSZ_TOPRIGHT – левый верхний/правый верхний угол. lParam – указатель на структуру RECT с определенными координатами экрана. Чтобы изменить размер или положение

прямоугольника перетаскивания, приложение должно изменить элементы этой структуры. Возвращаемое значение: если удалось обработать это сообщение, то TRUE.

Уведомительное сообщение **WM_SYSCHAR** посылается в очередь окна с фокусом клавиатуры, когда сообщение WM_SYSKEYDOWN(нажата F10 или ALT, затем другая клавиша) оттранслировано функцией TranslateMessage (переводит сообщения виртуальных клавиш в символьные сообщения.). Оно устанавливает символьный код системной символьной клавиши - то есть символьной клавиши, которая обрабатывается в то время, когда клавиша ALT находится в нажатом состоянии. Параметр: wParam - символьный код клавиши меню окна. lParam – счетчик повторов, код сканирования, флаг расширенного ключа, контекстный код, флаг предыдущего состояния ключа и флаг переходного состояния (например: 16,23 – скан-код, 29 – код контекста равен 1 если нажат ALT). Возвращаемое значение: приложение должно вернуть 0, после обработки этого сообщения.

Уведомительное сообщение **WM_SYSKEYUP** посылается окну с фокусом клавиатуры тогда, когда пользователь отпускает клавишу, которая была нажата, пока клавиша ALT удерживалась нажатой. Прикладная программа должна возвратить нуль, если она обрабатывает это сообщение. Параметр: wParam: виртуальный код освобождаемого ключа, lParam: lParam – счетчик повторов, код сканирования, флаг расширенного ключа, контекстный код, флаг предыдущего состояния ключа и флаг переходного состояния. Возвращаемое значение: приложение должно вернуть 0, после обработки этого сообщения.

Уведомительное сообщение **WM_LBUTTONDOWN** – посылается, если пользователь нажимает левую кнопку мыши, в то время, когда курсор находится в рабочей области окна. Если мышь не захвачена, сообщение посылается в окно под курсором. Иначе сообщение посылается в окно, которое захватило мышь. Возвращаемое значение Если приложение обрабатывает это сообщение, оно должно возвратить нуль. Параметр: wParam - указывает, находятся ли в нажатом состоянии различные виртуальные клавиши. Например:

MK_LBUTTON Левая кнопка мыши находится в нажатом состоянии.

MK_MBUTTON Средняя кнопка мыши находится в нажатом состоянии.

MK_RBUTTON Правая кнопка мыши находится в нажатом состоянии

Параметр lParam - младшее слово устанавливает x-координату курсора. Координата - относительно левого верхнего угла рабочей области. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области.: если приложение обрабатывает это сообщение, это должно возвратить нуль.

Уведомительное сообщение **WM_LBUTTONUP** – посылается, если пользователь отпускает левую кнопку мыши, в то время, когда курсор находится в рабочей области окна. Если мышь не захвачена, сообщение посылается в окно под курсором. Иначе сообщение посылается в окно, которое захватило мышь. Параметр wParam и lParam идентичны указанным в **WM_LBUTTONDOWN.** Возвращаемое значение: если приложение обрабатывает это сообщение, оно должно возвратить нуль.

Уведомительное сообщение **WM_LBUTTONDBLCLK** – посылается тогда, если пользователь дважды щелкает левой кнопкой мыши, в то время, когда курсор

находится в рабочей области окна. Если мышь не захвачена, сообщение посылается в окно под курсором. В противном случае, сообщение посылается в окно, которое захватило мышь. Параметр wParam и lParam идентичны указанным в **WM_LBUTTONDOWN.** Возвращаемое значение: если приложение обрабатывает это сообщение, оно должно возвратить нуль.

Только окна, которые имеют стиль CS_DBLCLKS, могут получать сообщения WM_LBUTTONDBLCLK, которые, система создает всякий раз, когда пользователь нажимает, отпускает и снова нажимает левую кнопку мыши в пределах системного ограничения времени двойного щелчка. Двойной щелчок левой кнопкой мыши фактически создает последовательность из четырех сообщений: WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDBLCLK и WM_LBUTTONUP.

Уведомительное сообщение **WM_MOUSEWHEEL** – отправляется в окно с фокусом, когда прокручивается колесико мыши. Параметр wParam – старшее слово указывает интервал, на который прокрутилось колесико, выраженный в нескольких или раздельных WHEEL_DELTA, число которых - 120. Положительное значение указывает, что колесико вращалось вперед, в сторону от пользователя; отрицательное значение указывает, что колесико вращалось назад, к пользователю. Младшее слово указывает, находятся ли в нажатом состоянии различные виртуальные клавиши (значения: MK_LBUTTON, MK_MBUTTON, MK_SHIFT, и т.д.). lParam аналогичен указанному в **WM_LBUTTONDBLCLK.** Возвращаемое значение: если приложение обрабатывает это сообщение, оно должно возвратить нуль.

4. С учётом описанной выше информации, изученной в процессе выполнения лабораторной работы, было создано приложение, поддерживающее описанные выше диалоговые окна, флаги, и сообщения, и т.п.

Листинг 2.1. Дескрипторы и объявления функций и переменных

```
#include <Windows.h>
#include <WindowsX.h>
#include <CommCtrl.h>
#include <tchar.h>
#include <fstream>
#include <string>
#include <codecvt>
#include <comdef.h> //для bstr_t
#include "resource.h"
#define LEFT 300
#define TOP  240

#pragma warning(disable : 4996) //отключает Ошибку deprecate. Возникает, когда используется
устаревшая функция
#define BEEP_TIMER 1
#define WM_ADDITEM WM_USER+1
#define BOLDDAY(ds, iDay)  \
        if (iDay > 0 && iDay < 32)(ds) |= (0x00000001 << (iDay - 1)) //for process the
MCN_GETDAYSTATE notification code

#pragma region Объявления
       #pragma region Handle of Window
       HWND hWnd = NULL; //дескриптор окна
       HWND hDlg = NULL; //декприптор диалогового окна
       HWND hSaveDlg = NULL; //save as
       HWND hRlsDlg = NULL; //replace
       #pragma endregion
       #pragma region Handle of control elements
              HWND hwndMonthCal = NULL; //для календаря
              HACCEL hAccel = NULL; //дескриптор акселератора
```

```
	#pragma endregion
	#pragma region For FINDREPLACE dialog
		FINDREPLACE findDlg; // структура для диалогового окна "Найти"
		UINT uFindMsgString = 0; // код сообщения FINDMSGSTRING
		HWND hFindDlg = NULL;
		/*Для FINDREPLACE*/
		TCHAR szBuffer[100] = TEXT("");
		TCHAR szBuffer1[100] = TEXT("");
		void OnFindMsgString(HWND hwnd, LPFINDREPLACE lpFindReplace); //Поиск, замена в
listbox
	#pragma endregion

	#pragma region MainWindowFunction
	//Здесь функции, переменные, структуры и т.п. для работы с главным окном приложения

	LRESULT CALLBACK MainWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

	BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCRStr);//создание окна и элементов на нем
	void OnCommand(HWND hWnd, int id, HWND hwnCTRL, UINT codeNotify);//обработка сообщений от
кнопок и SAVEAS

	//Обработка событий и сообщений мыши
	void OnLbuttonDClick(HWND hwnd, BOOL fDoubleClick, int x, int y, UINT keyFlags);//для
WM_LBUTTONDBLCLK и WM_LBUTTONDOWN
	void OnLButtonUp(HWND hwnd, int x, int y, UINT keyFlags);
	void MouseWheel(HWND hwnd, int xPos, int yPos, int zDelta, UINT fwKeys);

	//Обработка нажатия событий и сообщений клавиатуры
	void OnSysKey(HWND hwnd, UINT vk, BOOL fDown, int cRepeat, UINT flags);//для WM_SYSKEYUP

	/*Для изменения цвета окна, когда было обработано WM_SYSCHAR */
	RECT rc;
	HBRUSH brushes[3]; //кисти для изменения цвета окна
	int brush_index = 0;

	// обработка сообщения WM_TIMER
	void OnTimer(HWND hwnd, UINT id);//SetTimer, KillTimer, запуск зведного неба
	static int sec = 0; //записывается время работы с программой
	char workTime[10];//записывается время из sec для демонстрации пользователю

	SYSTEMTIME st;//структура для определения времени и даты
	void OnSizing(HWND hwnd, LPRECT lpRect, WPARAM wParam); //обработка onSizing

	//SAVEFILE AS dialog обработка ID_SAVE_AS
	OPENFILENAME OpenFDLG;//для создания диалогового окна Сохранить как
	UINT uOpenMSG = 0;

	#pragma endregion

	#pragma region EditDialogFunction
	//Здесь функции, переменные и т.п. для работы с диалоговым окном, добавляющим строки в
listbox
		INT_PTR CALLBACK DialogProc(HWND hWndlg, UINT uMsg, WPARAM wParam, LPARAM lParam);
		BOOL Dialog_OnInitDialog(HWND hWnd, HWND hWndF, LPARAM lParam);
		void OnAddItem(HWND hWnd);
		void Dialog_OnClose(HWND hWnd);
	#pragma endregion
	#pragma region  DialogProcMany
	//Здесь функции, переменные и т.п. если необходимо добавить более чем 1 запись в listbox*/
			void DialogMany_OnClose(HWND hWnd);
			BOOL Dialog_OnInitDialogMany(HWND hWnd, HWND hWndF, LPARAM lParam);
			INT_PTR CALLBACK DialogProcMany(HWND hWndlg, UINT uMsg, WPARAM wParam, LPARAM
lParam);

	#pragma endregion
#pragma endregion
```

Листинг 2.2. Функция создания главного окна приложения

```
LRESULT CALLBACK MainWindowProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
	{
		HDC hdc;			// индекс контекста устройства
		PAINTSTRUCT ps;		// структура для рисования

		switch (msg)
		{
			HANDLE_MSG(hwnd, WM_CREATE, OnCreate);
			HANDLE_MSG(hwnd, WM_COMMAND, OnCommand);
			HANDLE_MSG(hwnd, WM_LBUTTONDBLCLK, OnLbuttonDClick);
			HANDLE_MSG(hwnd, WM_LBUTTONDOWN, OnLbuttonDClick);
			HANDLE_MSG(hwnd, WM_LBUTTONUP, OnLButtonUp);
			HANDLE_MSG(hwnd, WM_SYSKEYDOWN, OnSysKey);
			HANDLE_MSG(hwnd, WM_SYSKEYUP, OnSysKey);
			HANDLE_MSG(hwnd, WM_KEYDOWN, OnSysKey);//чтобы поймать стрелки
			HANDLE_MSG(hwnd, WM_KEYUP, OnSysKey);//чтобы поймать стрелки
			HANDLE_MSG(hwnd, WM_MOUSEWHEEL, MouseWheel);

		case WM_TIMER:
		{
			OnTimer(hwnd,0);
			sec++;
			// В ответ на сообщение таймера выдаем
			 // звуковой сигнал
			MessageBeep(-1);
			return 0;
		}
			case WM_ADDITEM:
			{
				OnAddItem(hwnd);
				return 0;
			}

			case WM_SIZING:
			{
				OnSizing(hwnd, (LPRECT)lParam, (WPARAM)wParam);
				return 0;
			}

			case WM_SYSCHAR:
			{
				//ввод символов в программу
				/*ALT+<ANY KEY> перекрашивает экран одним из 3-х цветов*/
				wchar_t x = wParam;
				HWND hwndCtl = GetDlgItem(hwnd, IDC_LIST1);
				int count = ListBox_GetCount(hwndCtl);
				for (int i = 0; i < count; i++)
				{
					int len = ListBox_GetTextLen(hwndCtl, i);
					wchar_t* buffer = new wchar_t[len + 1];
					ListBox_GetText(hwndCtl, i, buffer);
					/*Найдет первое где встречается введенный символ*/
					for (int j = 0; j < len + 1; j++)
					{
						if (wcschr(buffer,x))
						{
							ListBox_SetCurSel(hwndCtl, i);
							if (brush_index == 2)
							brush_index = 0;
							else brush_index++;
							InvalidateRect(hwnd, NULL, FALSE);
							break;
						}
						else j++;
```

```
                                        }
                                        delete[]buffer;
                                }
                                /*Здесь ошибка не возникает
                                int len = ListBox_GetTextLen(hwndCtl, 0);
                                char* buffer = new char[len + 1];
                                delete[]buffer;*/
                                return 0;
                        }
                        case WM_PAINT:
                        {
                                PAINTSTRUCT ps;
                                BeginPaint(hwnd, &ps);
                                FillRect(ps.hdc, &ps.rcPaint, brushes[brush_index]);
                                EndPaint(hwnd, &ps);
                                return 0;
                        }

                        case WM_DESTROY:
                        {
                                KillTimer(hwnd, 1);
                                KillTimer(hwnd, BEEP_TIMER);
                                itoa(sec, workTime, 10); //преобразование подсчитанных секунд в
символы
                                MessageBoxA(NULL,  (LPSTR)workTime,  "Время  работы  программы
(сек.):", MB_ICONASTERISK | MB_OK);
                                PostQuitMessage(0);//without this app contie work after close
messagebox
                                return 0;
                        }

                        case WM_NOTIFY:
                        {
                                switch (((LPNMHDR)lParam)->code)
                                {
                                case MCN_GETDAYSTATE:
                                {
                                        /*выделяет 1  и 15 день месяца*/
                                //*https://docs.microsoft.com/ru-ru/windows/win32/controls/set-
day-states */
                                        MONTHDAYSTATE rgMonths[12] = { 0 };
                                        int cMonths = ((NMDAYSTATE*)lParam)->cDayState;
                                        for (int i = 0; i < cMonths; i++)
                                        {
                                                BOLDDAY(rgMonths[i], 1);
                                                BOLDDAY(rgMonths[i], 15);
                                        }
                                        ((NMDAYSTATE*)lParam)->prgDayState = rgMonths;
                                        return TRUE;
                                }//MCN_GETDAYSTATE
                                case MCN_SELECT:
                                {
                                        TCHAR szText[100];
                                        MonthCal_GetSelRange(GetDlgItem(hwnd,
IDC_MONTHCALENDAR1), &st);
                                        GetDateFormat(LOCALE_USER_DEFAULT,  DATE_LONGDATE,  &st,
NULL, szText, _countof(szText));
                                        HWND hwndCtl = GetDlgItem(hwnd, IDC_LIST1);
                                        ListBox_AddString(hwndCtl, szText);//добавляем выделенную
дату в список
                                        return TRUE;
                                }//MCN_SELECT
                                }
                        }

                        }
                if (uFindMsgString == msg)
```

```
		{
			OnFindMsgString(hwnd, (LPFINDREPLACE)lParam);
			return 0;
		}
		return (DefWindowProc(hwnd, msg, wParam, lParam));
	}


	BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCRStr)
	{

		CreateWindowEx(0, TEXT("ListBox"), NULL, WS_CHILD | WS_VISIBLE | WS_BORDER|
LBS_WANTKEYBOARDINPUT| LBS_NOTIFY, 10, 10, 250, 410, hwnd, (HMENU)IDC_LIST1, lpCRStr->hInstance,
NULL);
		hwndMonthCal = CreateWindowEx(0, TEXT("SysMonthCal32"), NULL, WS_CHILD |
WS_VISIBLE|WS_BORDER|    MCS_DAYSTATE|    MCS_MULTISELECT,    163,    31,    140,    100,    hwnd,
(HMENU)IDC_MONTHCALENDAR1, lpCRStr->hInstance, NULL);
		// Return if creating the month calendar failed.
		if (hwndMonthCal == NULL) return HRESULT_FROM_WIN32(GetLastError());
		// Get the size required to show an entire month.
		MonthCal_GetMinReqRect(hwndMonthCal, &rc);
		// Resize the control now that the size values have been obtained.
		SetWindowPos(hwndMonthCal, NULL, LEFT, TOP,    rc.right,          rc.bottom,
SWP_NOZORDER);

		MonthCal_SetCurrentView(hwndMonthCal, MCMV_YEAR);// Set the calendar to the
annual view.



		CreateWindowEx(0,   TEXT("Button"),   TEXT("ДОБАВИТЬ   ЗАПИСЬ"),   WS_CHILD   |
WS_VISIBLE | BS_PUSHBUTTON, 270, 10, 200, 40, hwnd, (HMENU)ID_NEW_RECORD, lpCRStr->hInstance, NULL);
		// создаём кнопку "Добавить неск. записей"
		CreateWindowEx(0,   TEXT("Button"),   TEXT("ДОБАВИТЬ   ЗАПИСИ"),   WS_CHILD   |
WS_VISIBLE | BS_PUSHBUTTON, 270, 55, 200, 40, hwnd, (HMENU)ID_NEW_RECORD2, lpCRStr->hInstance,
NULL);
		CreateWindowEx(0,   TEXT("Button"),   TEXT("УДАЛИТЬ    ЗАПИСЬ"),   WS_CHILD   |
WS_VISIBLE | BS_PUSHBUTTON, 270, 100, 200, 40, hwnd, (HMENU)ID_DEL_RECORD, lpCRStr->hInstance,
NULL);
		CreateWindowEx(0, TEXT("Button"), TEXT("ЗАМЕНИТЬ ЗАПИСЬ"),   WS_CHILD       |
WS_VISIBLE | BS_PUSHBUTTON, 270, 145, 200, 40, hwnd, (HMENU)ID_REPLACE, lpCRStr->hInstance, NULL);

		SetTimer(hwnd, BEEP_TIMER, 1000, NULL);
		return TRUE;
	}

	void OnTimer(HWND hwnd, UINT id)
	{
		int x, y;
		HDC hdc;
			hdc = GetDC(hwnd); //Выделяем контекст отображения
			x = rand() % 1600; //случайное число по x от 0 до 500
			y = rand() % 900; //по y от 0 до 400
			SetPixel(hdc, x, y, RGB(255, 255, 255)); //Выводим точку
			ReleaseDC(hwnd, hdc);
	}

	void OnCommand(HWND hWnd, int id, HWND hwnCTRL, UINT codeNotify)
	{
		HINSTANCE hInstance = GetWindowInstance(hWnd);

		switch (id)
		{
		case ID_NEW_RECORD:
		{
			int         DialogResult         =         DialogBoxParam(hInstance,
MAKEINTRESOURCE(IDD_DIALOG1), hWnd, (DialogProc), NULL);
```

```
                        if (IDOK == DialogResult)
                        {
                                SendMessage(hWnd, WM_ADDITEM, 0, 0);
                        }
                } break;
                case ID_NEW_RECORD2:
                {
                        int          DialogResult          =          DialogBoxParam(hInstance,
MAKEINTRESOURCE(IDD_DIALOG2), hWnd, (DialogProcMany), NULL);
                        if (IDOK == DialogResult)
                        {
                                SendMessage(hWnd, WM_ADDITEM, 0, 0);
                        }
                }
                break;
                case ID_DEL_RECORD: // нажата кнопка "Удалить запись"
                {
                        HWND hwndCtl = GetDlgItem(hWnd, IDC_LIST1);    //  получим  дескриптор
списка
                        int iItem = ListBox_GetCurSel(hwndCtl);// определим текущий выделенный
элемент в списке
                        if (iItem != -1)
                        {
                                int   mbResult   =   MessageBox(hWnd,   TEXT("Удалить   выбранный
элемент?"), TEXT("LW_OS_2"), MB_YESNO | MB_ICONQUESTION);

                                if (mbResult == IDYES)
                                {
                                        ListBox_DeleteString(hwndCtl,      iItem);//      удаляем
выделенный элемент из списка
                                }
                        }
                }
                break;
                case ID_SAVE_AS:
                {
                        char pListBox[] = { GetListBoxInfo(hWnd) };
                        TCHAR szFileName[MAX_PATH] = TEXT("");
                        OpenFDLG.lStructSize = sizeof(OPENFILENAME);
                        OpenFDLG.hInstance = GetWindowInstance(hWnd);
                        OpenFDLG.lpstrFilter   =   TEXT("Bitmape   Files(*.bmp)\0*.bmp\0   JPEG
files(*.jpg)\0*.jpg\0TEXT files(*.txt)\0*.txt\0All files(*.*)\0*.*\0\0");
                        OpenFDLG.lpstrFile = szFileName;
                        OpenFDLG.nMaxFile = _countof(szFileName);
                        OpenFDLG.lpstrTitle = TEXT("Сохранить как");
                        OpenFDLG.Flags = OFN_EXPLORER | OFN_ENABLESIZING | OFN_CREATEPROMPT |
OFN_OVERWRITEPROMPT | OFN_EXTENSIONDIFFERENT;
                        OpenFDLG.lpstrDefExt = TEXT("txt");

                        if (GetSaveFileName(&OpenFDLG) != FALSE)
                        {
                                MessageBox(hWnd,  szFileName,  TEXT("Сохранить  как"),  MB_OK  |
MB_ICONINFORMATION);
                                std::wofstream  fout(szFileName);  //  создаём  объект  класса
ofstream для записи
                                fout << "Файл создан в программе LWOS\n";
                                /*Соберем данные из listbox и добавим их в файл*/
                                HWND hwndCtl = GetDlgItem(hWnd, IDC_LIST1);
                                int count = ListBox_GetCount(hwndCtl);
                                for (int i = 0; i < count; i++)
                                {
                                        int len = ListBox_GetTextLen(hwndCtl, i);
                                        wchar_t* buffer = new wchar_t[len + 1];
                                        ListBox_GetText(hwndCtl, i, buffer);
                                        fout << buffer << "\n";
                                        delete[]buffer;
                                }
```

```
                                fout.close();

                        }
                    }break;
                    case ID_REPLACE: // нажата кнопка "Найти запись"
                        if (0 == uFindMsgString)
                        {
                            uFindMsgString = RegisterWindowMessage(FINDMSGSTRING);// получим
код сообщения FINDMSGSTRING
                        }
                        if (IsWindow(hFindDlg) == FALSE)
                        {
                            findDlg.lStructSize = sizeof(FINDREPLACE);
                            findDlg.hInstance = hInstance;// указываем дескриптор экземпляра
приложения
                            findDlg.hwndOwner = hWnd;               // указываем дескриптор
окна владельца
                            findDlg.lpstrFindWhat = szBuffer;
                            findDlg.lpstrReplaceWith = szBuffer1;
                            findDlg.wReplaceWithLen = _countof(szBuffer1);              //
указываем размер буфера
                            findDlg.wFindWhatLen = _countof(szBuffer);
                            hFindDlg = ReplaceText(&findDlg);
                        }
                        break;
                    }
                }
                    void OnLbuttonDClick(HWND hwnd, BOOL fDoubleClick, int x, int y, UINT keyFlags)
                    {
                        // передаём фокус клавиатуры главному окну
                        SetFocus(hwnd);

                        if (fDoubleClick == FALSE) // двойной клик
                        {
                            if (IsMaximized(hwnd)) // окно развёрнуто
                            {
                                ShowWindow(hwnd, SW_RESTORE);
                            }
                            else
                            {
                                // развернём окно
                                ShowWindow(hwnd, SW_MAXIMIZE);
                            }
                        }

                    }
                    void OnLButtonUp(HWND hwnd, int x, int y, UINT keyFlags)
                    {
                        ReleaseCapture(); // освободим мышь
                    }
                    void OnSysKey(HWND hwnd, UINT vk, BOOL fDown, int cRepeat, UINT flags)
                    {
                        RECT rect; // размеры окна

                // получим размеры окна
                        GetWindowRect(hwnd, &rect);

                        if (fDown)
                        {
                            switch (vk)
                            {
                            case VK_LEFT: // нажата стрелка влево
                                // перемещаем окно влево
                                SetWindowPos(hwnd, NULL, rect.left - 10, rect.top, 0, 0,
SWP_NOSIZE);
                                break;
```

```
                                case VK_RIGHT: // нажата стрелка вправо
                                        // перемещаем окно вправо
                                        SetWindowPos(hwnd, NULL, rect.left + 10, rect.top, 0, 0,
SWP_NOSIZE);
                                        break;

                                case VK_UP: // нажата стрелка вверх
                                        // перемещаем окно вверх
                                        SetWindowPos(hwnd, NULL, rect.left, rect.top - 10, 0, 0,
SWP_NOSIZE);
                                        break;

                                case VK_DOWN: // нажата стрелка вниз
                                        // перемещаем окно вниз
                                        SetWindowPos(hwnd, NULL, rect.left, rect.top + 10, 0, 0,
SWP_NOSIZE);
                                        break;
                                } // switch
                        } // if
                        else
                        {
                                switch (vk)
                                {
                                case VK_LEFT: // отпущена стрелка влево
                                        // перемещаем окно к левому краю
                                        SetWindowPos(hwnd, NULL, 0, rect.top, 0, 0, SWP_NOSIZE);
                                        break;

                                case VK_UP: // отпущена стрелка вверх
                                        // перемещаем окно к верхнему краю
                                        SetWindowPos(hwnd, NULL, rect.left, 0, 0, 0, SWP_NOSIZE);
                                        break;
                                }
                        }
                }
                void OnSizing(HWND hwnd, LPRECT lpRect, WPARAM wParam)
                {
                        /*изменение одной стороны ведет за собой увеличение другой*/
                        static double fixedRate = 1.0;
                        /*LPRECT lpRect = (LPRECT)lParam;*/
                        int w = lpRect->right - lpRect->left;//ширина
                        int h = lpRect->bottom - lpRect->top;//высота
                        int dw = (int)(h * fixedRate - w);//диагональ при изменении ширины
                        int dh = (int)(w / fixedRate - h);//диагональ при изменении высоты
                        switch (wParam)
                        {
                        case WMSZ_TOP:
                        case WMSZ_BOTTOM:
                                lpRect->right += dw;
                                break;
                        case WMSZ_LEFT:
                        case WMSZ_RIGHT:
                                lpRect->bottom += dh;
                                break;
                        case WMSZ_TOPLEFT:
                                if (dw > 0) lpRect->left -= dw;
                                else lpRect->top -= dh;
                                break;
                        case WMSZ_TOPRIGHT:
                                if (dw > 0) lpRect->right += dw;
                                else lpRect->top -= dh;
                                break;
                        case WMSZ_BOTTOMLEFT:
                                if (dw > 0) lpRect->left -= dw;
                                else lpRect->bottom += dh;
                                break;
                        }
```

```
                    }      // OnSizing
                    void MouseWheel(HWND hwnd, int xPos, int yPos, int zDelta, UINT fwKeys)
                    {
                           if (fwKeys & MK_SHIFT) // нажата клавиша SHIFT
                           {
                                  RECT rect; // размеры окна

                                  // получим размеры окна
                                  GetWindowRect(hwnd, &rect);

                                  // изменим размеры

                                  int d = 5 * zDelta / WHEEL_DELTA;
                                  InflateRect(&rect, d, d);

                                  MoveWindow(hwnd, rect.left, rect.top, (rect.right - rect.left),
(rect.bottom - rect.top), TRUE);
                           } // if
                    } // OnMouseWheel
```

Листинг 2.4. Определение функций для работы с Common Dialog

```
#pragma region For FINDREPLACE dialog
void OnFindMsgString(HWND hwnd, LPFINDREPLACE lpFindReplace)
              {
                    HWND hwndCtl = GetDlgItem(hwnd, IDC_LIST1);
                    if (lpFindReplace->Flags&FR_FINDNEXT) // нажата кнопка "Найти далее"
                    {

                           int iItem = ListBox_GetCurSel(hwndCtl);// определим текущий выделенный
элемент в списке
                           iItem = ListBox_FindString(hwndCtl, iItem, lpFindReplace-
>lpstrFindWhat);// выполним поиск указанного текста в списке
                           ListBox_SetCurSel(hwndCtl, iItem);// выделяем найденный элемент
                           if (LB_ERR == iItem) // элемент не найден
                           {
                                  MessageBox(hFindDlg, TEXT("Элемент не найден"), TEXT("LWOS"),
MB_OK | MB_ICONINFORMATION);
                           }

                    }
                    else
                    {
                           if (lpFindReplace->Flags&FR_REPLACE) // нажата кнопка "Заменить"
                           {

                                  int iItem = ListBox_GetCurSel(hwndCtl);
                                  iItem = ListBox_FindString(hwndCtl, iItem, lpFindReplace-
>lpstrFindWhat);
                                  ListBox_SetCurSel(hwndCtl, iItem);
                                  if (LB_ERR == iItem) // элемент не найден
                                  {
                                         MessageBox(hFindDlg, TEXT("Элемент не найден"),
TEXT("LWOS"), MB_OK | MB_ICONINFORMATION);
                                  }
                                  if (iItem != -1)
                                  {
                                         ListBox_DeleteString(hwndCtl, iItem);// удаляем
выделенный элемент из списка
                                         ListBox_InsertString(hwndCtl, iItem, lpFindReplace-
>lpstrReplaceWith); //добавление на то же место
                                         SetForegroundWindow(hwndCtl);
                                  }
                           }
                           else
                           if (lpFindReplace->Flags&FR_REPLACEALL)
                                  {
                                         int iItem = -1;
```

```
                                        for (;;)
                                        {
                                                int iItem = ListBox_GetCurSel(hwndCtl);
                                                iItem = ListBox_FindString(hwndCtl, iItem,
lpFindReplace->lpstrFindWhat);
                                                ListBox_SetCurSel(hwndCtl, iItem);//выделяем
элемент
                                                if (LB_ERR == iItem) // элемент не найден
                                                {
                                                        MessageBox(hFindDlg, TEXT("Элемент не
найден"), TEXT("LWOS"), MB_OK | MB_ICONINFORMATION);
                                                        break;
                                                }
                                                if (iItem != -1)
                                                {
                                                        ListBox_DeleteString(hwndCtl, iItem);//
удаляем выделенный элемент из списка
                                                        ListBox_InsertString(hwndCtl, iItem,
lpFindReplace->lpstrReplaceWith); //добавление на то же место
                                                        SetForegroundWindow(hwndCtl);
                                                }
                                        }//for
                                }//if
                        }
                }#pragma endregion
```

Листинг 2.5. Определение функций для работы с диалоговым окном для ввода текста

```
#pragma region EditDialogFunction
        INT_PTR CALLBACK DialogProc(HWND hWndlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
        {
                switch (uMsg)
                {
                        case WM_INITDIALOG:
                        {
                                BOOL bRet = HANDLE_WM_INITDIALOG(hWndlg, wParam, lParam,
Dialog_OnInitDialog);
                                return SetDlgMsgResult(hWndlg, uMsg, bRet);
                        }break;
                        case WM_COMMAND:
                        {
                                switch (LOWORD(wParam))
                                {
                                        case IDOK:
                                        {
                                                int cch = GetDlgItemText(hWndlg, IDC_EDIT1, szBuffer,
_countof(szBuffer));
                                                SetDlgItemText(hWndlg, IDC_EDIT1, NULL);
                                                SendMessage(GetParent(hWndlg), WM_ADDITEM, 0, 0);
//отправка текста с сообщением
                                        } break;
                                        case IDCANCEL:
                                        {
                                                EndDialog(hWndlg, 0);
                                                return FALSE;
                                        }break;
                                }
                        } break;
                        case WM_CLOSE:
                        {
                                HANDLE_WM_CLOSE(hWndlg, wParam, lParam, Dialog_OnClose);
                        }
                }
                        return FALSE;
        }

        BOOL Dialog_OnInitDialog(HWND hWnd, HWND hWndF, LPARAM lParam)
        {
                HWND hwndEdit = GetDlgItem(hWnd, IDC_EDIT1);
```

```
		Edit_LimitText(hwndEdit, _countof(szBuffer)-1);
		Edit_SetCueBannerText(hwndEdit, L"Новая запись");
		return TRUE;
	}
	void OnAddItem(HWND hWnd)
	{
		HWND hWNDctrl = GetDlgItem(hWnd, IDC_LIST1);
		int iItem = ListBox_AddString(hWNDctrl, szBuffer);
		ListBox_SetCurSel(hWNDctrl, iItem);
	}
	void Dialog_OnClose(HWND hWnd)
	{
		if (hWnd == hDlg) { DestroyWindow(hWnd); }
		else { EndDialog(hWnd, 0); }
	}
#pragma endregion
```

Листинг 2.5. Определение функций для множественного добавления записей

```
#pragma region ManyStringAdd
	INT_PTR CALLBACK DialogProcMany(HWND hWndlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
	{
		switch (uMsg)
		{
		case WM_INITDIALOG:
		{
			BOOL bRet = HANDLE_WM_INITDIALOG(hWndlg, wParam, lParam,
Dialog_OnInitDialogMany);
			return SetDlgMsgResult(hWndlg, uMsg, bRet);
		}break;
		case WM_COMMAND:
		{
			TCHAR str[] = TEXT("");
			switch (LOWORD(wParam))
			{
			case IDOK:
			{
				int counter = GetDlgItemInt(hWndlg,IDC_EDIT2,NULL,NULL);
				int cch = GetDlgItemText(hWndlg, IDC_EDIT3, szBuffer,
_countof(szBuffer));
				while (counter != 0)
				{
				SetDlgItemText(hWndlg, IDC_EDIT3, NULL);
				SendMessage(GetParent(hWndlg), WM_ADDITEM, 0, 0);
				counter--;
				}

			} break;
			case IDCANCEL:
			{
				EndDialog(hWndlg, 0);
				return FALSE;
			}break;
			}
		} break;
		case WM_CLOSE:
		{
			HANDLE_WM_CLOSE(hWndlg, wParam, lParam, DialogMany_OnClose);
			}
		}
		return FALSE;
	}
	BOOL Dialog_OnInitDialogMany(HWND hWnd, HWND hWndF, LPARAM lParam)
	{
		HWND hwndEdit = GetDlgItem(hWnd, IDC_EDIT3);
		SendDlgItemMessage(hwndEdit, IDC_SPIN1, UDM_SETRANGE, 0, 2);
		Edit_LimitText(hwndEdit, _countof(szBuffer) - 1);
		Edit_SetCueBannerText(hwndEdit, L"Новая запись");
		return TRUE;
```

```
}
void DialogMany_OnClose(HWND hWnd)
{
	if (hWnd == hDlg) { DestroyWindow(hWnd); }
	else { EndDialog(hWnd, 0); }
}
 #pragma endregion
```

5. Тестирование:

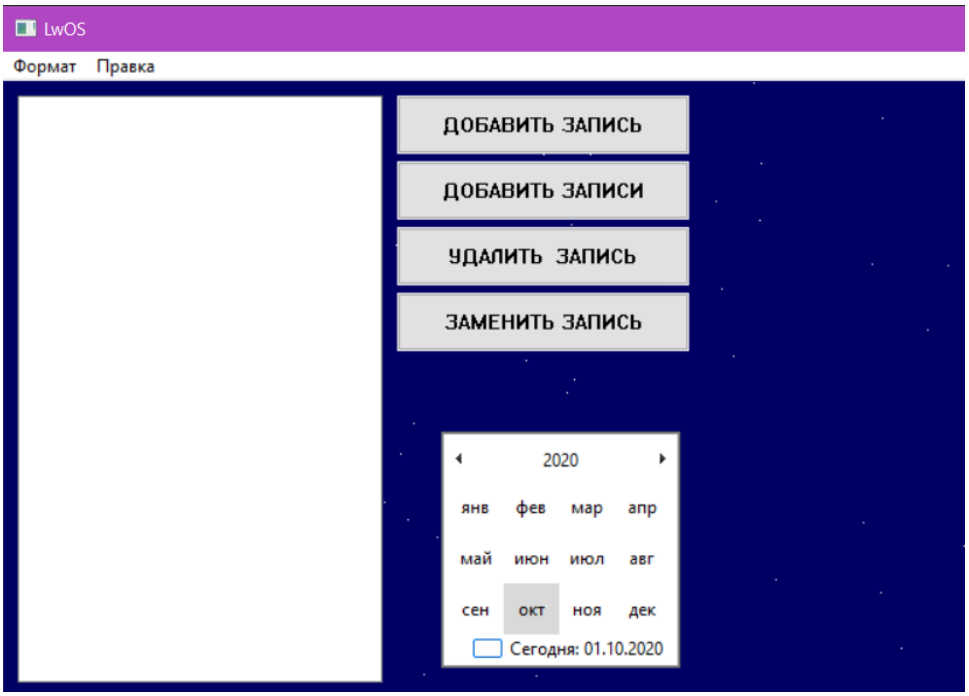

Рис. 2.3. общий вид приложения

Добавление одной записи в listbox становится возможным после нажатия на кнопку «Добавить запись». Если необходимо добавить несколько одинаковых записей, то нужно нажать на кнопку «Добавить записи». В поле со счетчиком вводится количество добавляемых записей.

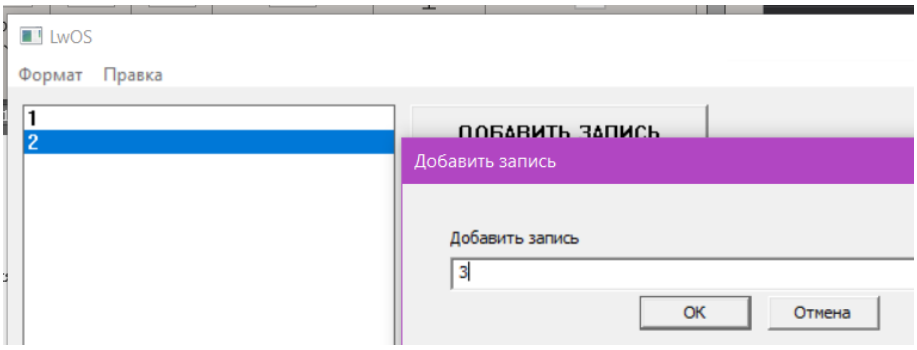

Рис. 2.4. Добавление записей в listbox

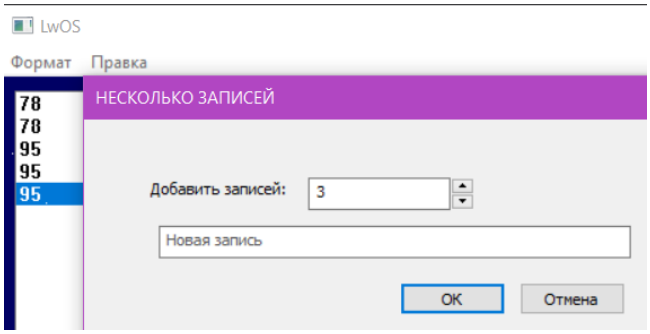

Рис. 2.5. Добавление нескольких записей

Чтобы удалить одну из записей необходимо выделить ее в списке и затем нажать на кнопку «Удалить запись».

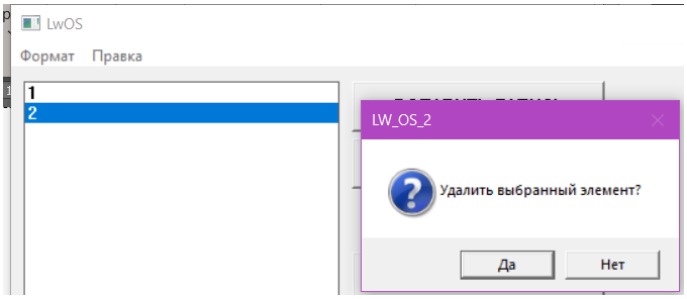

Рис. 2.6. Удаление записи

Замена одной или нескольких записей осуществляется через окно «Заменить» и открывается или при нажатии кнопки «Заменить запись» или сочетания клавиш CTRL+H

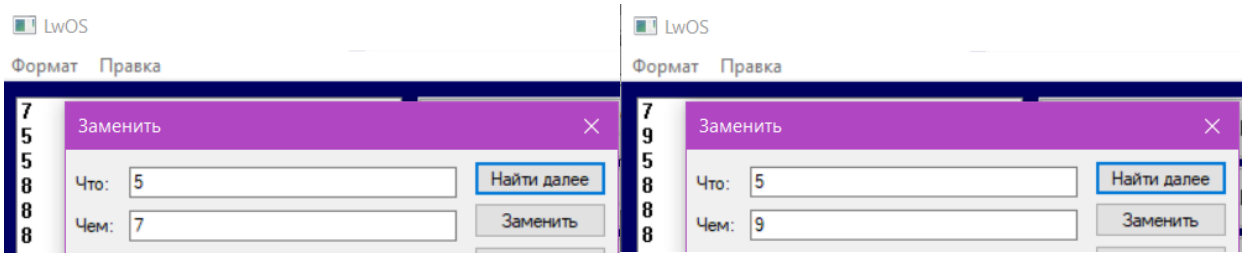

Рис. 2.7. Диалоговое окно «Заменить»

При нажатии сочетания клавиш CTRL+S открывается диалоговое окно для сохранения данных из listbox.

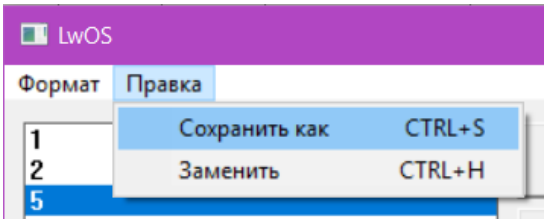

Рис. 2.8. Меню

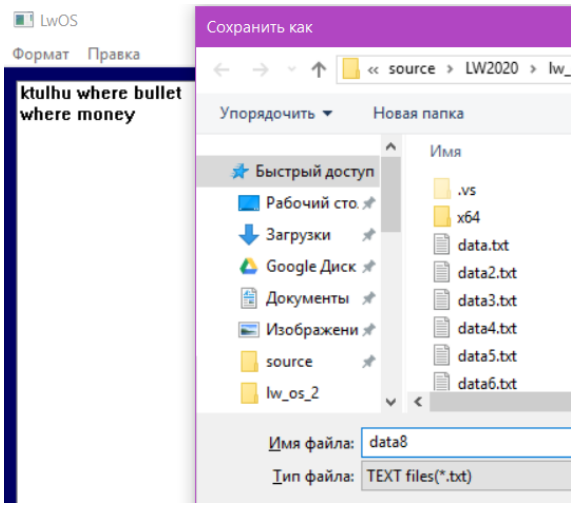

Рис. 2.9. Диалоговое окно «Сохранить как»

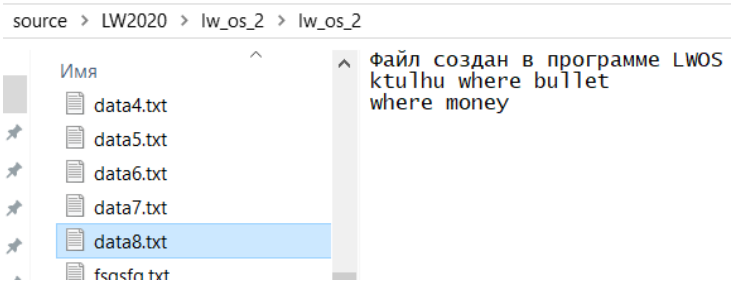

Рис. 2.10. Результат работы программы

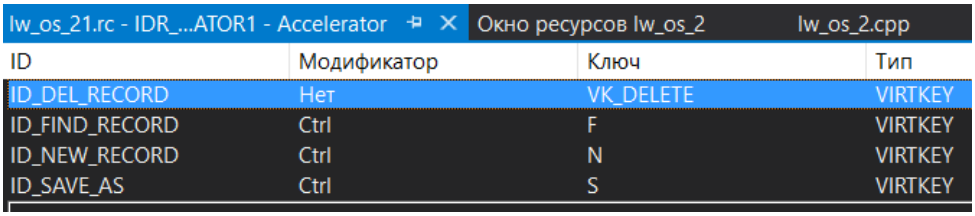

Рис. 2.11. Таблица «быстрых клавиш»

Обработка события WM_NOTIFY связана с уведомлением MCN_SELECT от календаря, которое отправляется, если в календаре пользователь выделяет какой-либо день. Например, на представленном ниже рисунке была выделена дата 17 июля 2020 года и при обработке событий была добавлена в listbox.

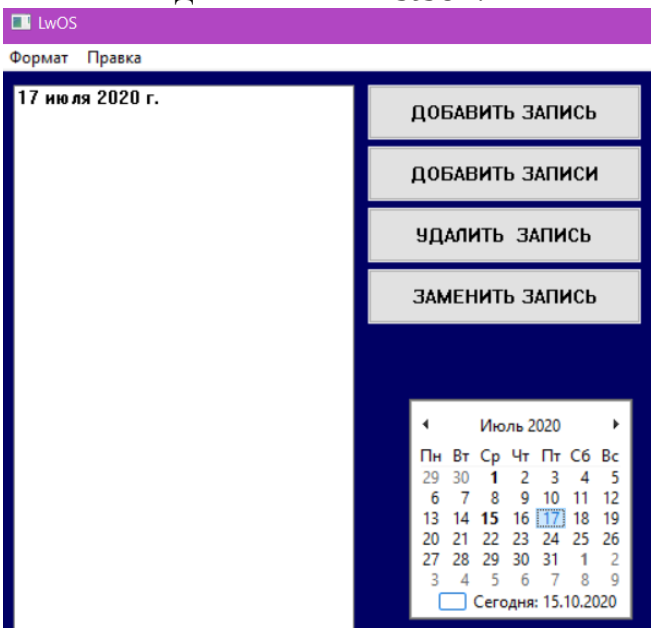

Рис. 2.12. Обработка WM_NOTIFY

В listbox можно осуществлять поиск строки, где встречается буква из сочетания клавиш: ALT+<Символ>. В примере: ALT+W и ALT+M.

Рис. 2.13. Демонстрация найденной строки

При запуске приложения устанавливается таймер. В процессе работы приложения функция главного окна примерно раз в 1 секунду получает сообщение WM_TIMER, при этом осуществляется подсчет времени работы приложения в секундах и сопровождается звуковым сигналом (от BEEP_TIMER).

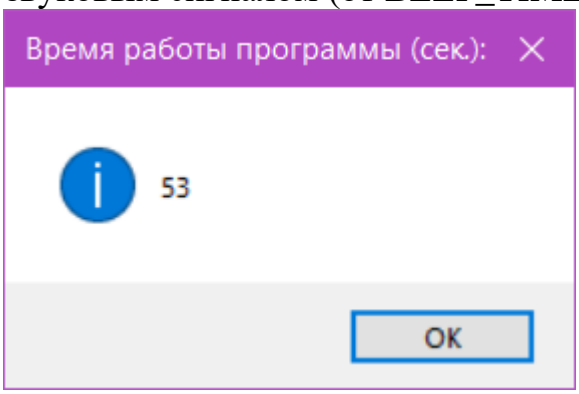

Рис. 2.14. Результат работы WM_TIMER

6. Возникшие ошибки: нарушение прав доступа при чтении при попытке открыть диалоговое окно «Сохранить как».

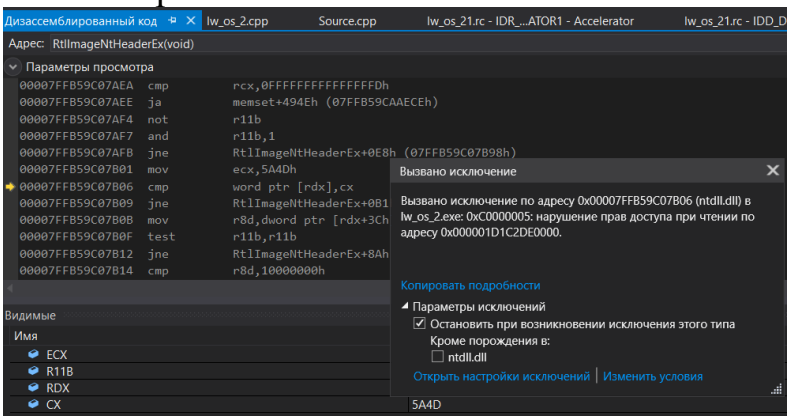

Рис. 2.15. Нарушение прав доступа при чтении

При открытии она «Заменить» существует возможность найти элемент в Listbox, только если обе строки содержат одинаковое значение. Если указать разные данные для замены (заменить значение 3 значением 5), то можно наблюдать, что в полях структуры lpFindReplace , связанных с этими строками содержится одинаковое значение, равное второй строке.

Решение: в каждую из строк передавать отдельные переменные szBuffer и szBuffer2.

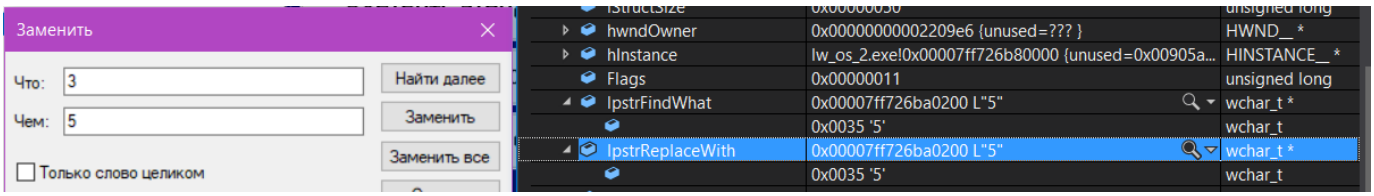

Рис. 2.16. Одно и то же значение в структуре

Вывод: Таким образом, в ходе выполнения лабораторной работы было осуществлено знакомство с процессом создания оконных приложений с применением WinApi. Были изучены элементы общего пользования, диалоговые окна. Было создано оконное приложение, в котором определен элемент Listbox для ввода информации. В него при нажатии на кнопку «Добавить запись» можно добавить

значение и удалить их при нажатии на «Удалить запись». Управление в приложении можно осуществлять и с помощью быстрых клавиш. Так, например сочетание клавиш CTRL+S откроет окно «Сохранить как», а CTRL+N вызовет диалоговое окно «Добавить запись».

Таким образом, используя Win API можно создать пользовательское приложение.