МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ”ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМУВАННЯ, ЧАСТИНА 2 (ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ)

МЕТОДИЧНІ ВКАЗІВКИ

до виконання розрахунково-графічної роботи
для студентів базового напряму ”Комп’ютерна інженерія”

Затверджено
на засіданні кафедри
”Електронно-обчислювальних машин”
Протокол № 8 від 27.03.2014 р.

Львів 2014

Програмування, частина 2 (Об'єктно-орієнтоване програмування): Методичні вказівки до виконання розрахунково-графічної роботи для студентів базового напряму "Комп'ютерна інженерія". Укл.: Ю.В.Морозов, М.В.Олексів – Львів: Видавництво Національного університету "Львівська політехніка", 2014. – 39 с.

Укладачі Морозов Ю.В., к.т.н., доцент
Олексів М. В., к.т.н., асистент

Відповідальний за випуск: Мельник А. О., професор, завідувач кафедри

Рецензент Дунець Р.Б., д.т.н., проф., зав. каф. СКС

РОЗРОБКА ДІАЛОГОВИХ ПРОГРАМ ЗАСОБАМИ MFC

Мета: Оволодіти навиками розробки програм з графічними діалоговими інтерфейсами з використанням бібліотеки класів MFC.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Концепція роботи windows програм

Програма, що написана мовою C/C++, з консольним інтерфейсом працює від початку функції `main` до її кінця. Для здійснення операцій вводу/виводу програма викликає функції вводу/виводу операційної системи (ОС). ОС не викликає прикладну програму. *Windows програма з віконним інтерфейсом* працює навпаки. Програма тільки запускається операційною системою і все. Далі програма нічого не робить, а чекає поки не отримає повідомлення від ОС, яке потім буде опрацьовуватися. **Повідомлення – це сигнал про те, що відбулася деяка подія.** Будь-яка подія супроводжується повідомленнями, що надсилаються або конкретному вікну, кільком вікнам, чи всім вікнам одразу. Деякі події можуть породити ще кілька подій, наприклад створення вікна супроводжується його перемальовуванням. Прикладами події є переміщення курсору мишки, натискання кнопки, тощо.

На відміну від звичайних C/C++ програм, де операції в програмі виконувалися лінійно, в Windows програмах виконується опрацювання повідомлень, які з'являються в довільному порядку, тобто неочікувано (асинхронно). Цим Windows програма схожа на обробник переривань.

Повідомлення мають унікальний номер і часто мають символьні мнемонічні позначення, наприклад `WM_PAINT` замість `0x000F`, і два числові параметри. Список повідомлень є у файлі winuser.h. Сюди можна додати свої повідомлення з унікальними номерами. М

Повідомлення передається програмі від ОС через спеціальну функцію Windows. Після чого вони надходять в чергу повідомлень програми, яка об'являється наступним макросом в класі діалогу: `DECLARE_MESSAGE_MAP()`. При надходжені повідомлення з черги на обробку відбувається пошук його номера в карті повідомлень. При наявності номера повідомлення в карті повідомлень відбувається виклик зв'язаного з повідомленням обробника. Повідомлення, що відслідковуються і мають свої нестандартні обробники заносяться між макросами `BEGIN_MESSAGE_MAP` і `END_MESSAGE_MAP`.

Макрос обробника повідомлення має наступну структуру: тип повідомлення, наприклад `ON_BN_CLICKED, ON_COMMAND,....;` номер повідомлення; назва обробника події.

Наприклад:

```
BEGIN_MESSAGE_MAP(C***App, CWinApp)
      ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()

BEGIN_MESSAGE_MAP(CLogin, CDialog)
      ON_BN_CLICKED(IDOK, OnBnClickedOk)
END_MESSAGE_MAP()
```

Події можна генерувати вручну методами `SendMessage` – синхронний метод (чекає поки повідомлення не буде опрацьоване), `PostMessage` – асинхронний метод (не чекає на опрацювання повідомлення, а продовжує виконання програми).

Структура віконної діалогової MFC програми

Бібліотека Microsoft Foundation Classes (MFC) дає можливість розробляти GUI-застосунки для Microsoft Windows на мові C++ з використанням багатого набору бібліотечних класів. Велика частина MFC є відносно тонким об'єктно-орієнтованим шаром над Windows API. Це рішення, з одного боку, підвищує продуктивність, але, з другого боку, успадковує всі недоліки дизайну Windows API і перешкоджає перенесенню програм на інші платформи.

Типова діалогова програма, що написана з використанням бібліотеки класів MFC складається з двох основних класів: класу `C***App` і класу `C***Dlg`, де замість зірочок по замовчуванню мітиться назва проекту.

Клас **C***App** містить метод **`InitInstance()`**, з якого починається виконання програми і створення об'єкту програми **`theApp`**. В функції **`InitInstance()`** ініціюється черга повідомлень, створюється об'єкт діалогу, до нього приєднується вікно діалогу і діалог відображається на екрані. Початково діалог містить 2 кнопки "**OK**" і "**Cancel**". Кожна з них має свій обробник, який можна редагувати.

Клас **C***Dlg** містить функції **`OnInitDialog()`** та **`OnPaint()`**. В функції **`OnInitDialog()`** налаштовується зовнішній вигляд вікна діалогу та можна робити деякі початкові налаштування і присвоєння до моменту відображення діалогу. На момент виклику більшість об'єктів вже створено. Функція **`OnPaint()`** здійснює вивід вікна діалогу на екран. Кожен з класів містить карту повідомлень.

При створенні проекту діалогової програми у MS Visual Studio ці класи генеруються автоматично. Особливістю класу діалогу є те, що у нього інкапсулюються об'єкти, що прив'язані до контролів (кнопок, радіо кнопок, полів введеня тощо), меню, та інші об'єкти, що реалізують бізнес логіку програми.

Структура бібліотеки класів MFC

Структура бібліотеки класів MFC відображена на діаграмі класів (див. рис.1). В основі більшості класів бібліотеки MFC лежить базовий клас `CObject`. Як видно з рис. 1 клас, що забезпечує роботу з діалогами (`CDialog`), знаходиться в ієрархії класів `CObject` → `CCmdTarget` → `CWnd`. Отже, завдяки спадкуванню, клас діалогу `CDialog` володіє всією функціональність цих класів. Найціннішим у цій ієрархії є клас `CWnd`, який має множину методів, що забезпечують керування відображенням діалогу, обмін даними між контролами і змінними до яких вони прив'язані за допомогою механізму DDX/DDV; методи взаємодії з чергою повідомлень, меню, стандартними обробниками деяких подій, таймером. Базовий віконний клас `CWnd` описаний нижче.

Члени Дані

m_hWnd – містить хендл вікна (**HWND**), що відповідає вікну.

Функції стану вікна

GetActiveWindow – Повертає активне вікно.

GetFocus – Повертає об'єкт **CWnd,** що є у фокусі.

GetWindowContextHelpId – Повертає ідентифікатор контексту довідки.

ModifyStyle – Модифікує стиль поточного вікна.

SetWindowContextHelpId – Встановлює ідентифікатор контексту довідки.

Розмір і позиціонування вікна

CloseWindow – Мінімізує вікно.

GetClientRect – Повертає розміри клієнтської області вікна.

GetWindowRect – Повертає екранні координати вікна.

SetWindowPos – Changes Змінює розмір, розташування і порядок дочірніх, виринаючих (pop-up) вікон і вікон верхнього рівня.

SetWindowRgn – Встановлює область вікна.

Функції доступу до вікна

FindWindow – Повертає хендл вікна, яке ідентифікується за допомогою імені і класу вікна

GetDlgCtrlID – Якщо вікно **CWnd** є дочірнім, то виклик цього методу повертає його ідентифікатор.

GetDlgItem – Повертає контрол, що розташований у вікні діалозгу, якому відповідає ідентифікатор, що передається у метод через параметр.

UpdateData – Встановлює значення у контрол, або читає значення з контролу, який розташований на діалозі.

UpdateDialogControls – Викликаєтсья для оновлення стану контролів на діалозі.

Методи Оновлення/Перемалювання вікна

GetDC – Повертає контекст відображення клієнтської області вікна.

GetWindowDC - Повертає контекст відображення цілого вікна включаючи заголовок вікна, меню і елементи прокрутки (scroll bars).

Invalidate – Перемальовує (оновлює) клієнтську область.

ReleaseDC – Вивільняє контексти клієнтської області і вікна, забезпечуючи доступ іншим засобам.

ShowWindow – Показує/приховує вікно.

Функції перетворення координат

ClientToScreen – Перетворює координати точки або прямокутника з системи координат клієнтської області вікна у систему координат екрану.

MapWindowPoints – Перетворює множину точок з просторо координат поточного вікна у прості координат іншого вікна.

ScreenToClient – Перетворює екранні координати вказаної точки або прямокутника у систему координат клієнтської області вікна.

Текстові функції вікна

GetWindowText – Повертає текст, що міститься у вікні, або заголовок вікна (caption).

SetWindowText – Встановлює текст у вікні або заголовок вікна (caption).

GetFont – Повертає поточний стиль тексту.

SetFont – Встановлює поточний стиль тексту.

Функції взаємодіє з елементами діалогів

GetDlgItemInt – Перетворює текст, що міститься у контрлі на діалозі, у цілочисельне.

GetDlgItemText – Повертає заголовок, або текст, що асоціюється з контролом.

SendDlgItemMessage – Посилає повідомлення контролу на вікні.

Функції по роботі з меню

GetMenu – Повертає вказівник на меню діалогу.

SetMenu – Встановлює вказане меню як меню діалогу.

Функції по роботі з таймером

SetTimer – Встановлює системний таймер, що посилає при спрацюванні повідомлення WM_TIMER.

KillTimer – Усуває системний таймер.

Функції повідомлень

MessageBox – Створює і відображає вікно з вказаним повідомленням.

Фіунції керування повідомленнями вікна

PostMessage – Розміщує повідомлення у черзі повідомлень програми, після чого завершує свою роботу не чекаючи на опрацювання розміщеного повідомлення.

PreTranslateMessage - Використовується **CWinApp** для фільтрації повідомлень вікнам перед тим як вони надсилаються у віконні функції **TranslateMessage** і **DispatchMessage**.

SendMessage - Sends a message to the **CWnd** object and does not return until it has processed the message.

Перевизначаються

DoDataExchange – Використовується механізмом DDX/DDV. Викликається методом **UpdateData**.

WindowProc – Забезпечує віконну процедуру для даного вікна (Класично – саме віконна процедура здійснює керування опрацюванням повідомлень (визначає який метод яке повідомлення опрацьовує), але у MFC, по замовчуванню, керування опрацюванням повідомлень відбувається через карту повідомлень.

Команда довідки Обробники і Функції

OnHelp – Викликається при натисканні кнопки F1 (Довідка).

Обробники повідомлень ініціалізації

OnInitMenu – Викликається перед активацією меню.

OnInitMenuPopup - Викликається перед активацією виринаючого (pop-up) меню.

Загальні обробники повідомлень

OnClose – Викликається при надходжені сигналу закриття вікна.

OnCreate – Викликається при створені вікна.

OnCtlColor - Викликаєтсья коли поточне вікно є батьківським вікном контрола перед тим, як контрол буде перемальованим.

OnSize – Викликається коли розмір поточного вікна змінився.

Обробники повідомлень операцій введення

OnChar – Викликається коли натиснута несистемна клавіша.

OnLButtonDblClk – Викликається при подвійному натисканні лівої кнопки мишки.

OnLButtonDown - Викликається при натисканні лівої кнопки мишки.

OnLButtonUp - Викликається при відтисканні лівої кнопки мишки.

OnMouseMove – Викливкаєтсья коли курсор мишки перемістився.
OnMouseWheel – Викликається коли користувач прокрутив колесико мишки .
OnTimer – Викливкається при спрацюванні таймеру, що створений методом SetTimer.

Опертаори
operator HWND – Викливається щоб отримати хендл вікна.

Характеристика деяких класів бібліотеки MFC

`CEdit` – клас реалізує Edit Box, найпростіший текстовий редактор.
`CRichEditCtrl` - клас реалізує Rich Edit Control, текстовий редактор, що за своєю функціональністю схожий на wordpad.
`CStatic` – реалізує можливість розміщувати на діалозі статичний текст.
`CButton` – різні кнопочки: радіо-кнопочки, кнопочки, чекбокси.
`CListBox` – вікно з списком (текстовий список, де кожен елемент є з нового рядка, може містити в кожному з полів невидиму додаткову інформацію)
`CComboBox` – реалізує випадаючий список.
`CDC` – клас, що реалізує контексти пристроїв. Графічне розміщення інформації можливе переважно за допомогою маніпулювання класами пензлика, ручки, палітри в контексті пристрою. Контексти пристроїв є апаратно-залежними, тобто можуть відображати однакові значення кольорів в залежності від можливостей конкретного пристрою.
`CString` - клас реалізує текстові рядки. Клас описаний у MSDN в розділі *CStringT Class*. Методи класу `CString` описані в розділі MSDN присвяченому *CStringT Members*.

Меню

Меню в MFC реалізовується за допомогою класу `CMenu`. Цей клас інкапсулює в собі об'єкт Windows, що визначається дескриптором `HMENU`. Конструктор об'єкт класу `CMenu` створює екземпляр класу, **але не саме меню**! Після створення об'єкту треба використати спеціальні функції для його безпосереднього створення, або завантажити його з ресурсів.

Щоб отримати вказівник на меню треба викликати метод класу `CWnd::GetMenu()`. Приклад:

```
CWnd* pMain = AfxGetMainWnd();

// The main window _can_ be NULL, so this code
// doesn't ASSERT and actually tests.
if (pMain != NULL)
{
   // Get the main window's menu
   CMenu* pMenu = pMain->GetMenu();
....
}
```

далі меню можна редагувати програмно.

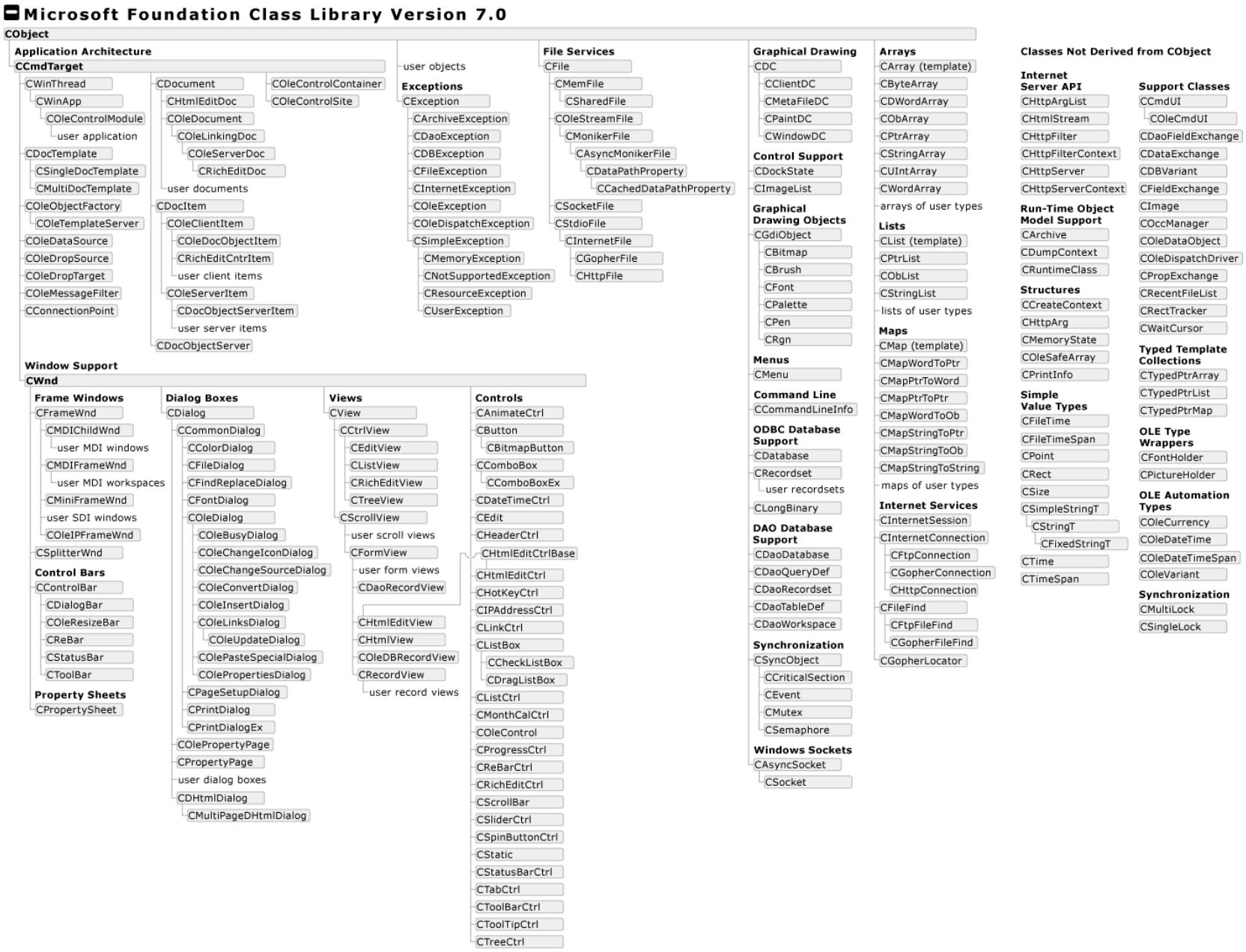

Рис. 1. Діаграма класів MFC

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке MFC?
2. Охарактеризуйте базовий віконний клас `CWnd`?
3. Які класи бібліотеки MFC ви знаєте?
4. Структура типової діалогової програми?
5. Що таке повідомлення?
6. Яке призначення макросу `DECLARE_MESSAGE_MAP`?
7. Що таке DDX/DDV?
8. Наведіть основні етапи написання програми з використанням MFC.

ЛІТЕРАТУРА

1. Олафсен Ю., MFC и Visual C++ 6. Энциклопедия программиста (+ CD) / Ю. Олафсен, К. Скрайбнер, К. Дэвид Уайт. – М.: DiaSoft, 2004. – 992 с.
2. Сидорина Т. Самоучитель Microsoft Visual Studio C++ и MFC (+ CD-ROM). – СпБ: BHV, 2009. – 848 с.
3. Давыдов В. Visual C++. Разработка Windows-приложений с помощью MFC и API-функций (+ CD-ROM). – СпБ: BHV, 2008. –576 с.
4. MSDN [електронний ресурс]. – Режим доступу до онлайн довідки: http://msdn.microsoft.com/en-us/

ЗАВДАННЯ

1. Написати віконну діалогову програму, яка записує і читає з файлової бази даних текстову інформацію про об’єкти, що описують предметну область задану варіантом. Програма має відповідати наступним вимогам:
    - а) Забезпечити перевірку на коректність введення даних за допомогою механізму виключних ситуацій. При спробі введення некоректних даних відобразити на екрані відповідне повідомлення за допомогою методу MessageBox та скасувати операцію.
    - б) Для введення даних використати елемент керування типу Edit Box, а вміст файлової бази даних відобразити у елементі керування типу List Box. Позначити призначення елементів керування за допомогою групуючи контролів (Group Box).
    - в) Продублювати функції кнопок запису/читання у меню.
    - г) Передбачити можливість вибору файлу, з яким відбуватиметься робота у процесі виконання програми за допомогою діалогу вибору файлу.
    - д) Шляхом спадкування від класу CStdioFile або CFile створити власний клас, який забезпечуватиме беферизований запис даних у файл (Запис даних спочатку здійснюється в програмний буфер. При спробі додати нову порцію даних у повний буфер спочатку має відбуватися запис всіх даних з буферу у файл, очищення буферу і лише тоді додавання цієї порції даних у порожній буфер). Буфер реалізувати за допомогою структури даних з бібліотеки STL визначеній варіантом. Розмір буферу визначається варіантом.
2. Скомпілювати та відлагодити програму.
3. Скласти звіт про виконану роботу з приведенням тексту програми та результату її виконання.
4. Дати відповідь на контрольні запитання.

ПРЕДМЕТНІ ОБЛАСТІ

Варіант	*Завдання*
1	Базовий клас: **class Device** **{** **public:** **Device(char* fName);** **~Device();** **virtual bool Open() = 0;** **virtual bool Close() = 0;** **virtual bool Execute(char* cmd, void* prm) = 0;** **virtual bool Status(int ext=0) {return isOpened;}** **protected:** **char* deviceName;** **char* friendlyName;** **bool isOpened;** **};** **Device()** – конструктор базового класу. Виділяє пам‘ять під змінну friendlyName та ініціалізує її. **~Device()** – деструктор базового класу. Вивільняє пам‘ять виділену під змінні deviceName (!якщо виділено!) та friendlyName. Друкує повідомлення якщо робота з пристроєм не була корректно завершена. **Open()** – відкриває пристрій для роботи. Виділяє пам‘ять та ініціалізує змінну deviceName, встановлює змінну isOpened. Друкує повідомлення, про те що пристрій готовий до роботи. **Close()** – завершує роботу з пристроєм. Друкує повідомлення, та встановлює змінну isOpened у відповідний стан. **Execute()** – виконує команду специфічну для кожного пристрою. **Status()** – повертає стан пристрою. Похідний клас **Printer.** Атрибути: **bool canPrint;** **char* ptrBuf;** Команди для функції **Execute()**: **“Print”** – друкує вмістиме буферу **“Write”** – завантажує текст у буфер (prm – розглядати як char*), змінює значення змінної canPrint. **“Clear”** – обнулює вмістиме буферу, змінює значення змінної canPrint + Перевизначити функцію Status() – коли (ext == 1) повертати значення (isOpened && canPrint). Визначити конструктор та деструктор (!вивільняти всі ресурси!) класу.
2	Базовий клас див. завд. 1 Похідний клас **Scaner**. Атрибути: **char* scrBuf;** Команди для функції **Execute()**: **“Scan”** – заповнити вмістиме буферу випадковими даними. **“Read”** – завантажує текст у prm – розглядати як char*. **“Clear”** – обнулює вмістиме буферу. Визначити конструктор та деструктор (!вивільняти всі ресурси!) класу.
3	Базовий клас:

	class Animal **{** **protected:** **char* name;** **int weight;** **public:** **Animal() ;** **virtual ~Animal() ;** **virtual void Call() = 0;** **virtual bool Feed();** **virtual void DoAnimalStuff();** **};** **Animal()** – конструктор базового класу. Виділяє пам‘ять під змінну name та ініціалізує змінну weight = 0. **~Animal()** – деструктор базового класу. Вивільняє пам‘ять віділену під змінну name. **Call()** – імітує притаманні тварині звуки (!виводить на екран повідомлення!). **Feed()** – збільшує вагу тварини після годування (змінює змінну weight). **DoAnimalStuff()** – зменшує вагу тварини (змінює змінну weight) та виводить повідомлення. Похідні класи **Dog** та **Cat**. Конструктори похідних класів ініціалізують змінну name та weight. Перевизначити функції **Feed()** та **DoAnimalStuff().** В основній програмі створити кілька тварин і змоделювати їхню поведінку.
4	Базовий клас: **class Shape2D** **{** **public:** **Shape2D();** **virtual ~ Shape2D();** **virtual float Area( )= 0;** **virtual float Perimeter() = 0;** **virtual void PrintMessage();** **};** **Shape2D()** – конструктор базового класу. **~ Shape2D()** – деструктор базового класу. **Area()** – повертає значення площі фігури. **Perimeter()** – повертає значення периметру фігури. **PrintMessage()** – виводить повідомлення про тип фігури. Похідні класи **Triangle** та **Rectangle** Визначити необхідні для похідних класів параметри та перевизначити необхідні функції.
5	Базовий клас: **class Shape3D** **{** **public:** **Shape3D();** **virtual ~ Shape3D();**

	virtual float Area( )= 0; **virtual float Volume() = 0;** **virtual void PrintMessage();** **};** **Shape3D()** – конструктор базового класу. **~ Shape3D()** – деструктор базового класу. **Area()** – повертає значення площі фігури. **Volume ()** – повертає значення об'єму фігури. **PrintMessage()** – виводить повідомлення про тип фігури. Похідні класи **Sphere** та **Cube** Визначити необхідні для похідних класів параметри та перевизначити необхідні функції.
6	Базовий клас: **class Resource** **{** **protected:** **int id;** **bool isUsed;** **char* title;** **char* author;** **public:** **Resource(int number) ;** **virtual ~ Resource() ;** **virtual bool TakeResource ();** **virtual bool GiveBackResource();** **virtual void PrintInfo();** **virtual bool Status() {return isUsed;}** **};** **Resource()** – конструктор базового класу. Ініціалізує змінну id та isUsed. Виділяє пам'ять під **title** та **author**. **~ Resource()** – деструктор базового класу. **TakeResource()** – встановлює змінну isUsed=1 (виводить на екран повідомлення). **GiveBackResource()** – встановлює змінну isUsed=0 (виводить на екран повідомлення). **PrintInfo ()** – виводить інформацію про ресурс. **Status()** – повертає стан ресурсу. Похідний клас **Book**. Атрибути: **int pages;** **int year;** Похідний клас **CD**. Атрибути: **int cdType;** Визначити конструктор та деструктор (!вивільняти всі ресурси!) класу. В основній програмі створити декілька ресурсів та продемонструвати процес їхнього використання.

ВАРІАНТИ ЗАВДАННЬ

Варіант	№ предметної області	Структура даних	Розмір буферу
1	1	queue	5
2	2	vector	4
3	3	list	3
4	4	deque	6
5	5	map	4
6	6	multimap	5
7	1	vector	5
8	2	list	4
9	3	deque	3
10	4	map	5
11	5	multimap	4
12	6	queue	6
13	1	list	2
14	2	deque	3
15	3	map	4
16	4	multimap	3
17	5	queue	5
18	6	vector	3
19	1	deque	4
20	2	map	5
21	3	multimap	4
22	4	queue	3
23	5	vector	4
24	6	list	5
25	1	map	6
26	2	multimap	4
27	3	queue	5
28	4	vector	3
29	5	list	4
30	6	deque	5

ПОРЯДОК ВИКОНАННЯ

1. Запустити MS Visual Studio 2012

2. У меню середовища MS Visual Studio вибрати «Файл» → «Создать» → «Проект». Вибрати тип проекту «Visual C++» → «Приложение MFC», задати ім’я проекту RGR (див. рис. 2) і натиснути «ОК».

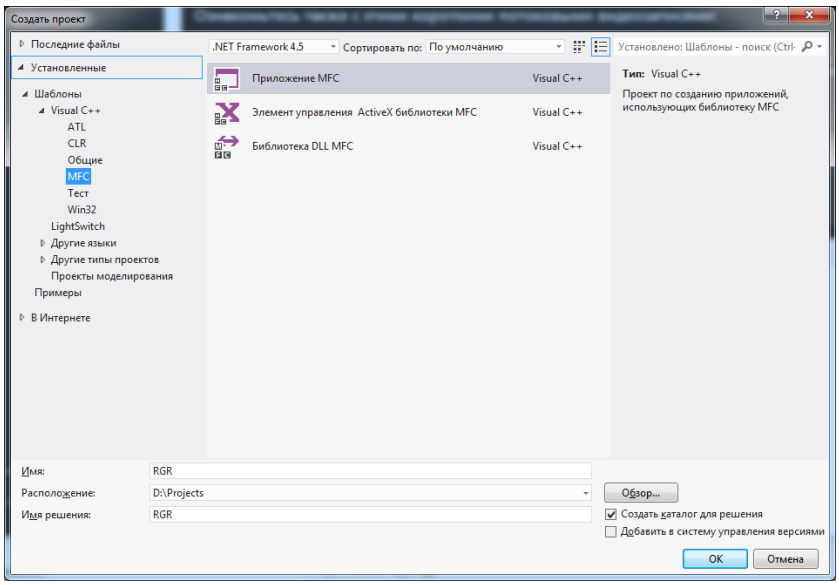

Рис.2. Створення проекту з використанням бібліотеки MFC.

3. У вікні, що відкриється вибрати пункт «Тип приложения» де вибрати «На основе диалоговых окон» та зняти пташку з «Использовать библиотеки с поддержкой Юникода» (див. рис. 3). Ознайомитися з вмістом інших вкладок. Натиснути «Готово».

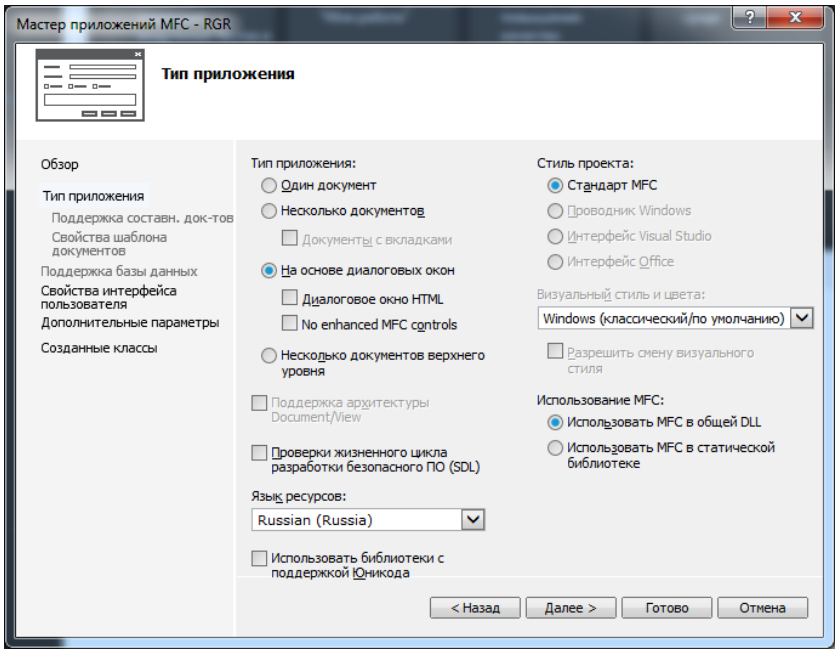

Рис.3. Налаштування властивостей проекту

В результаті цих дій має створитися проект типу діалог (див. рис. 4) який складається з таких файлів:

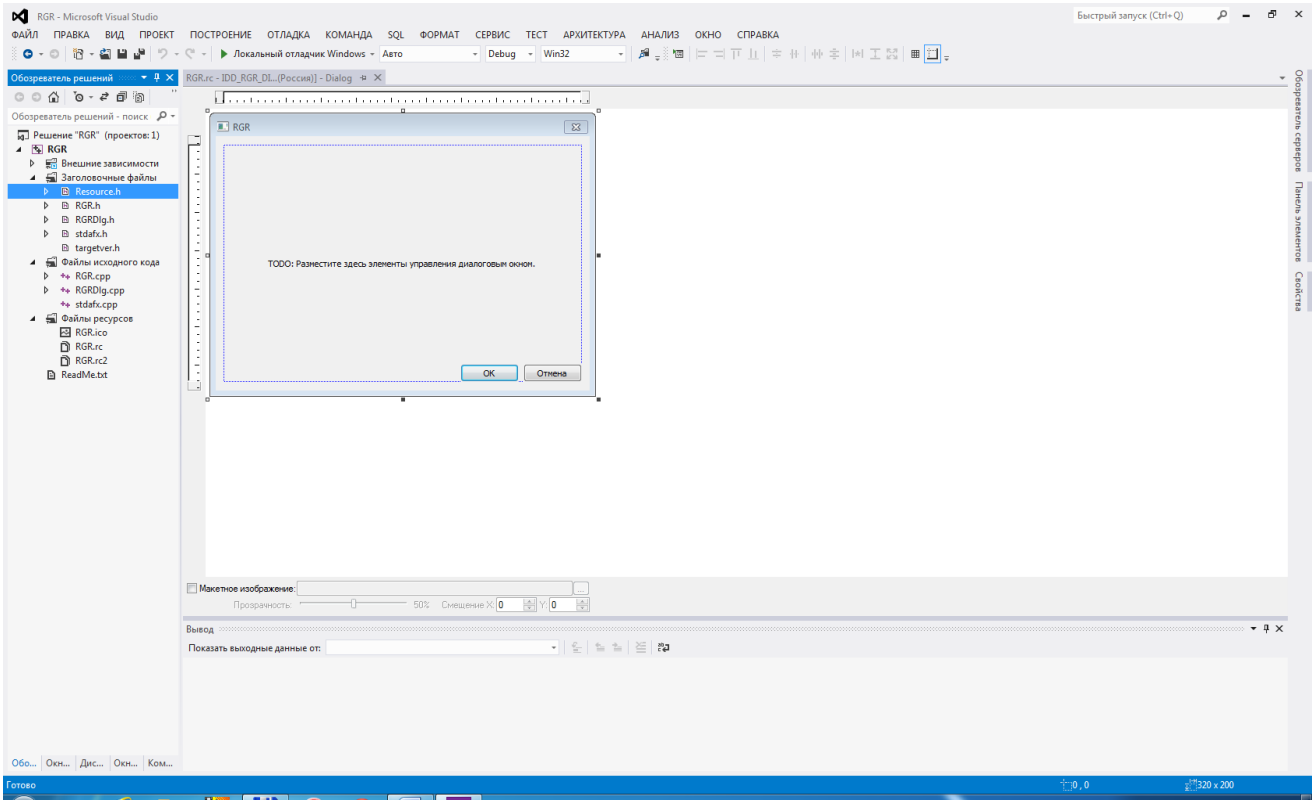

Рис.4. Вигляд проекту

- Resource.h – заголовочний файл, що описує ідентифікатори ресурсів.
- RGR.h – заголовочний файл, що описує клас задачі (CRGRApp).
- RGRDlg.h – заголовочний файл, що описує клас діалогу (CRGRDlg).
- Stdafx.h – наперед скомпільований заголовочний файл, що містить файли, що часто використовуються в проекті, але рідко змнюються. Використання цього файлу дозволяє зменшити час компіляції складних проектів.
- targetver.h – забезпечує визначення останньої доступної платформи Windows.
- RGR.cpp – файл, що містить визначення методів, що оголошені у класі CRGRApp.
- RGRDlg.cpp – файл, що містить визначення методів, що оголошені у класі CRGRDlg.
- Stdafx.cpp – тіло наперед скомпільованого заголовочного файлу RGR.pch.
- RGR.ico – файл з іконками програми.
- RGR.rc – файл з ресурсами, у якому місяться у графічному вигляді діалоги, меню, курсори, іконки, а також таблиця текстових рядків, що будуть доступні через ідентифікатори і інформація про версію програми. Ці ресурси можна редагувати у MS Visual Studio. Ресурси не можна відкрити поки є відкритий файл Resource.h.
- RGR.rc2 – файл з ресурсами, які не можна редагувати.
- ReadMe.txt – містить інформацію про призначення файлів проекту, що були згенеровані MS Visual Studio.

4. Додати до проекту класи з бізнес логікою. Для цього в меню середовища MS Visual Studio натиснути «Проект» → «Добавить класс…». У вікні, що відкрилося вибрати «Класс C++» (див. рис. 5) та натиснути «Добавить». У наступному вікні задати назву класу та інші його характеристики такі як базовий клас, наявність віртуального деструктора тощо (див. рис. 6). Натиснути «Готово». В результаті цих дій має утворитися клас з заданим у налаштуваннях ім’ям, що розміщується у двох файлах – заголовочному і файлі з реалізацією класу. Імена цих файлів задаються при створенні класу. При необхідності створити таким самим способом всі класи, що необхідні для реалізації бізнес логіки програми.

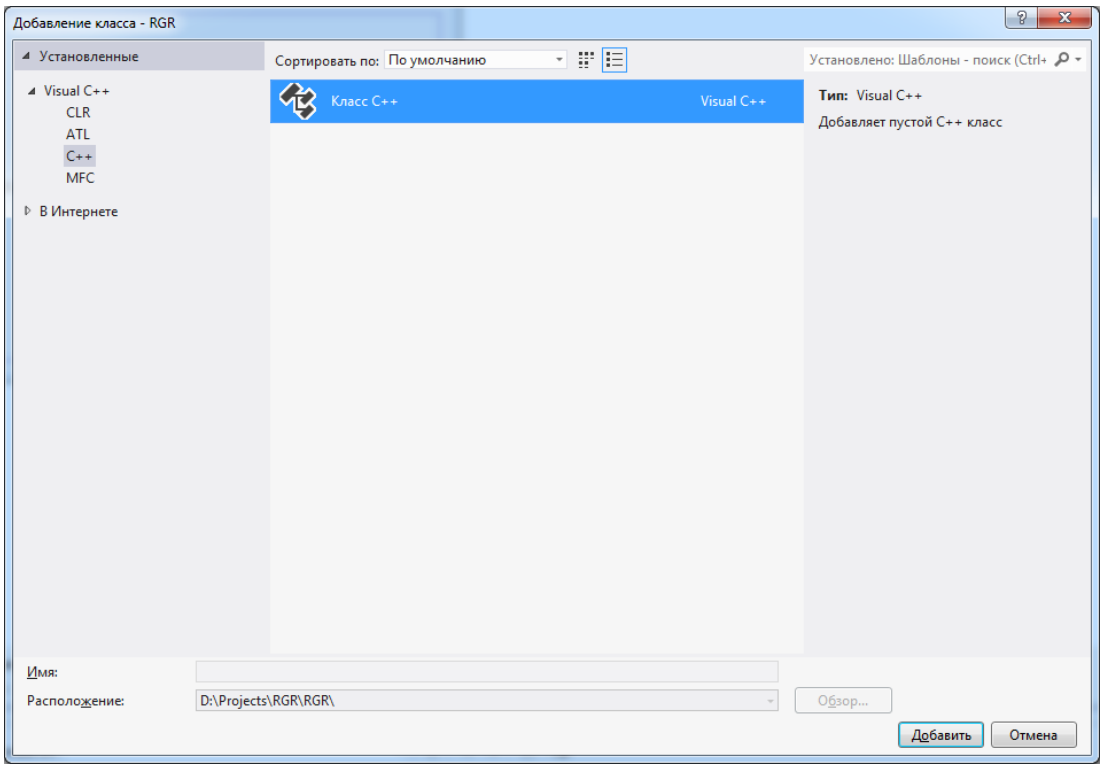

Рис. 5. Вибір типу класу, що додаватиметься

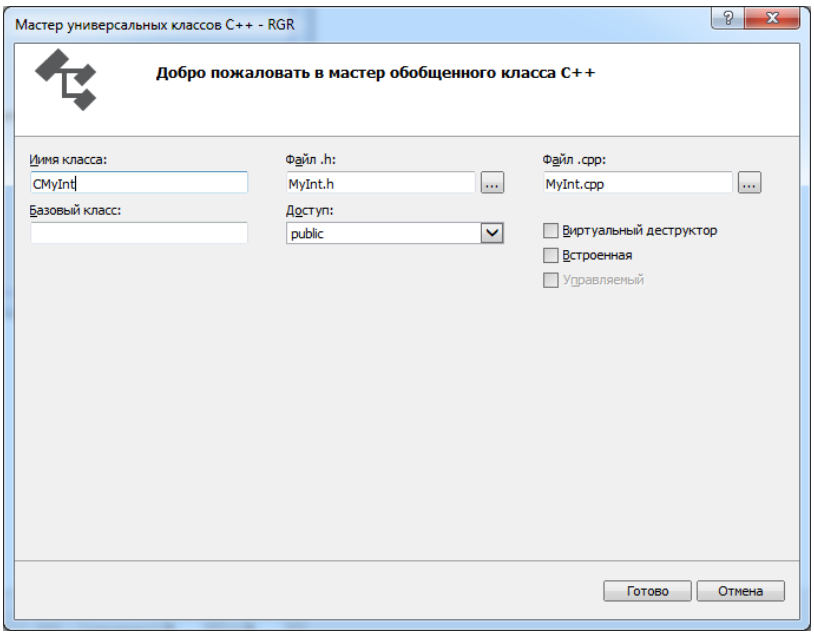

Рис. 6. Налаштування характеристик класу, що додаватиметься

У даному прикладі використовуватиметься клас `CMyInt` з наступним вмістом:

```
//---------------------------------------- Файл MyInt.h ---------------------------------
	#pragma once
	class CMyInt
	{
		int * m_pnNum;
	public:
		/*
		 *  При використанні динамічного виділення пам'яті у членів-даних класу
		 *  слід обов'язково перевизначити конструктор за замовчуванням,
```

```
         *  конструктор копіювання, деструктор та оператор присвоювання, які
         *  мають забезпечити коректне опрацювання
         *  динамічно створених членів класу
         */

        CMyInt(void);
        CMyInt(const CMyInt&);
        ~CMyInt(void);

        CMyInt& operator=(const CMyInt&);

        void SetInt(int);
        int GetInt(void) const;

};
```

//---------- Файл MyInt.cpp ----------

```
#include "stdafx.h"
#include "MyInt.h"


CMyInt::CMyInt(void)
{
        m_pnNum = new int;
        *m_pnNum = 0;
}

CMyInt::CMyInt(const CMyInt& obj)
{
        m_pnNum = new int;
        *m_pnNum = *obj.m_pnNum;
}

CMyInt::~CMyInt(void)
{
        delete m_pnNum;
}

CMyInt& CMyInt::operator=(const CMyInt& obj)
{
        if (this != &obj)
        {
                *m_pnNum = *obj.m_pnNum;
                return *this;
        }
        return *this;
}

void CMyInt::SetInt(int n)
{
        *m_pnNum = n;
}

int CMyInt::GetInt(void) const
{
        return *m_pnNum;
}
```

5. Інкапсулювати об’єкт необхідного створеного класу у клас `CRGRDlg` у який вводитимуться дані з елементів керування діалогу та об’єкт структури даних, що застосовуватиметься для буферизації:

```
#pragma once
#include "MyInt.h"
#include <list>
#include <iterator>
using namespace std;

// диалоговое окно CRGRDlg
class CRGRDlg : public CDialogEx
{
private:
	CMyInt m_intObj;
	list<int> lst;
	……
```

6. У ресурсах відкрити графічне зображення діалогу. Для цього у меню MS Visual Studio вибрати: «Вид» → «Ресурсы». У панелі, що відкрилася, двічі клікнути лівою кнопкою мишки на елемент `IDD_RGR_DIALOG`. В результаті цього має відкритися графічне зображення діалогу. Далі у меню MS Visual Studio вибрати: «Вид» → «Панель элементов». В результаті має відкритися панель, що містить типові елементи керування (контроли) (див. рис.7).

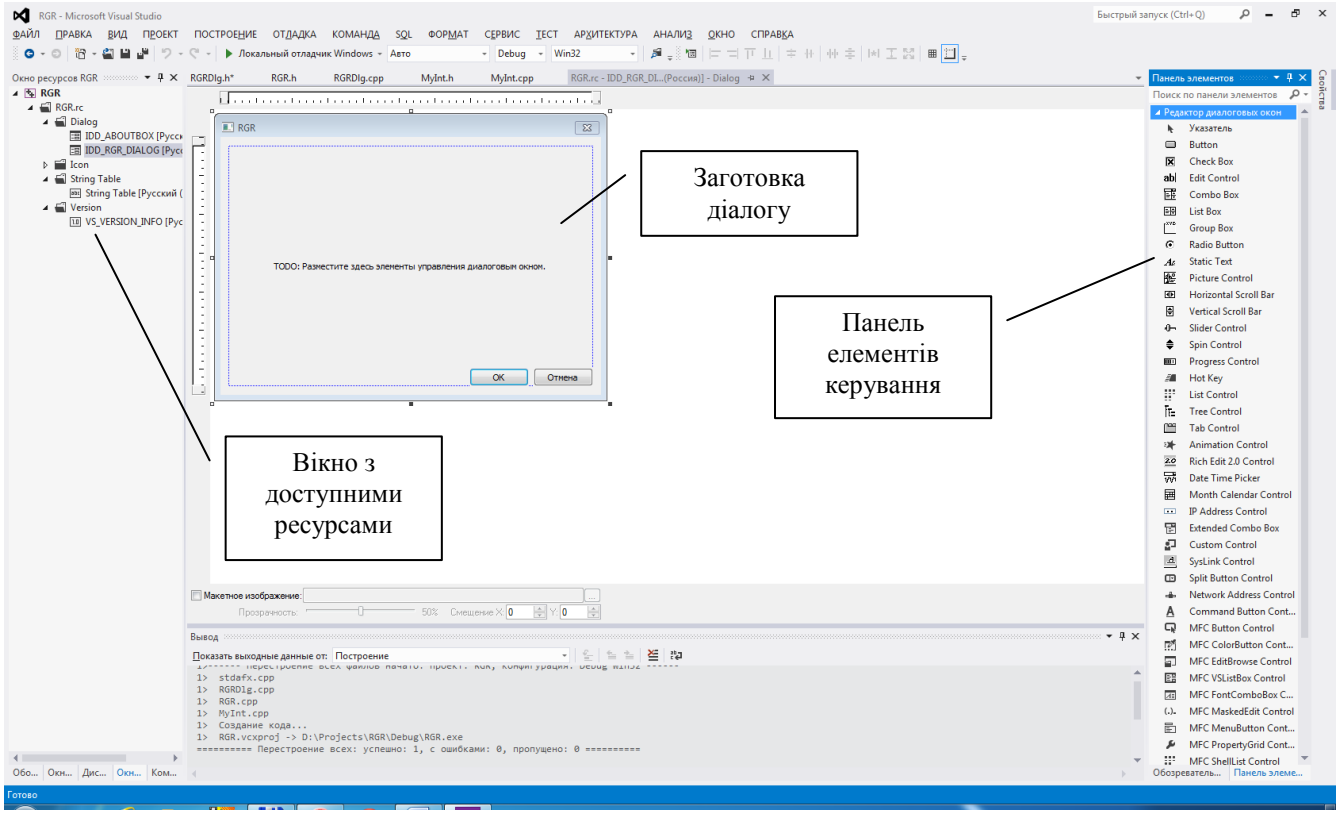

Рис.7. Редактор діалогів

7. У редакторі діалогів вибрати елемент типу Lable з текстом «`TODO:…`» та видалити його натисканням кнопки «Delete». Те саме повторити для двох кнопок: «ОК» і «Отмена». Вибрати в панелі елементів керування контрол «Edit Control» та перетянути його на діалог. Те саме повторити для двох елементів типу «Group Box», двох елементів типу «Button» та одного елементу «List Control». Розташувати їх так, як зображено на рис. 8.

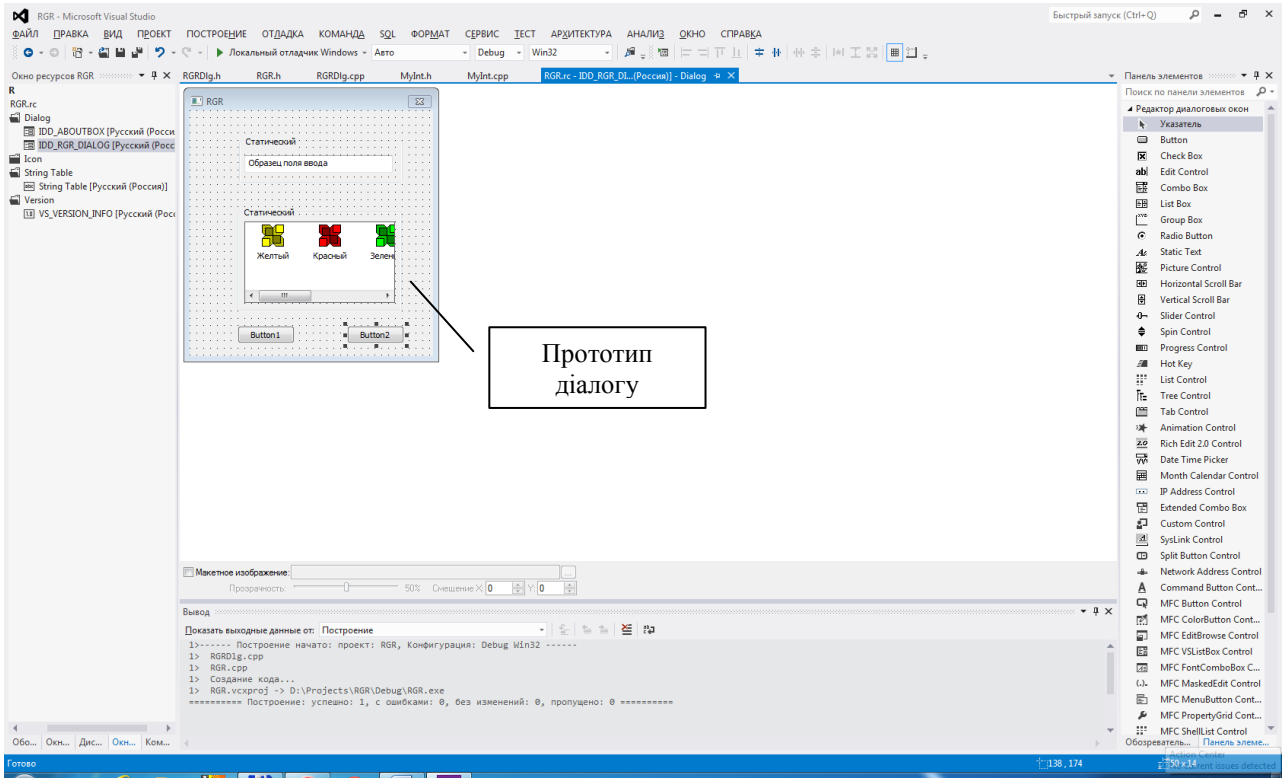

Рис.8. Редактор діалогів з створеним прототипом діалогу

8. Вибрати мишкою елемент керування типу «Group Box» навколо елемента керування типу «Edit Control». Клікнути на ньому правою кнопкою мишки та вибрати в випадаючому меню пункт «Свойства». В результаті відкриється панель властивостей елемента керування «Group Box» (див. рис. 9). Ознайомитися з її вмістом. Двічі клікнути мишкою на тексті «Статический» у полі «Подпись». Видалити текст «Подпись» і замість нього написати «Поле введення даних» та натиснути «Enter». Те саме повторити для іншого елемента керування «Group Box».

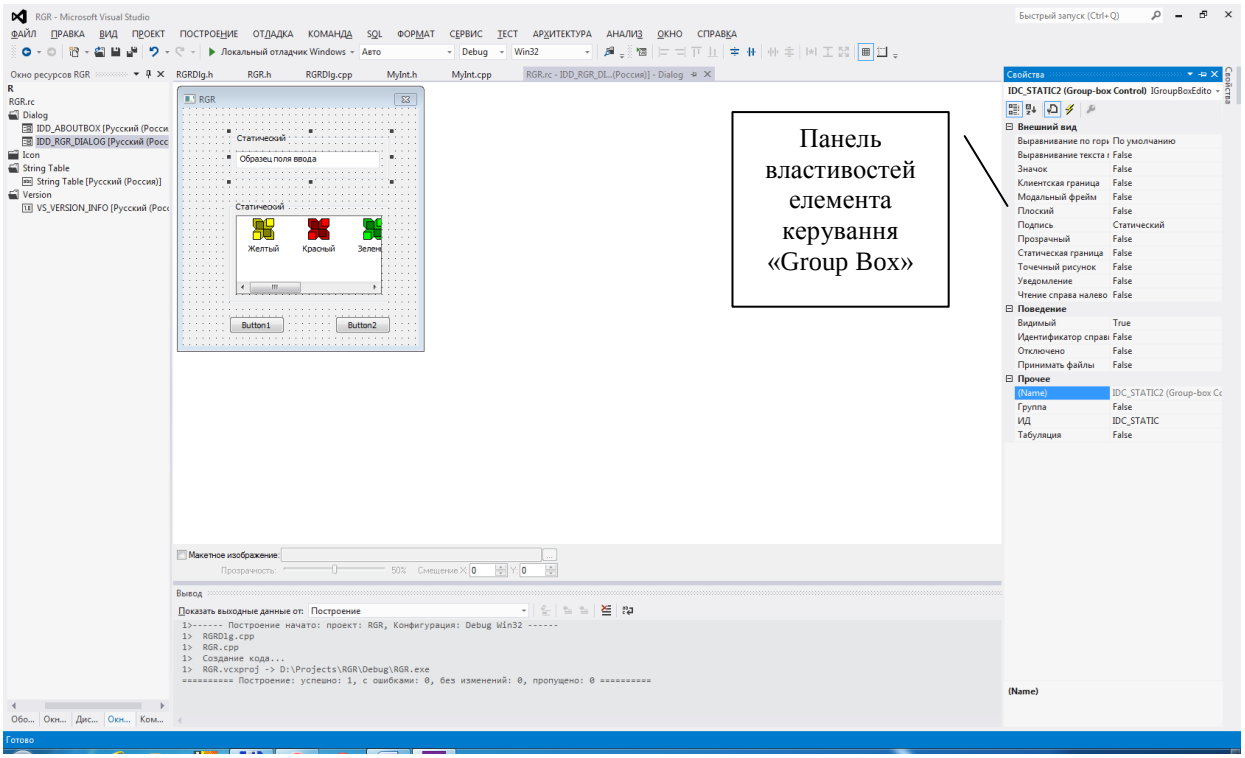

Рис. 9. Редактор діалогів з відкритою панеллю властивостей елемента керування «Group Box».

9. Вибрати елемент керування типу «Edit Control» та відкрити його властивості. Ознайомитися з його властивостями. Клікнути правою кнопкою мишки на елемент керування типу «Edit Control» та вибрати пункт меню «Добавить переменную». В результаті має відкритися вікно, що зображене на рис. 10. Задати значення полів так, як зображено на рис. 10. На даному рисунку вказується, що елемент керування з ідентифікатором `IDC_EDIT1` зв'язується за допомогою механізму DDX/DDV по значенню (value) зі змінною `m_nData` типу `int`, яка буде розташована у полі `private` класу `CRGRDlg`. Натиснути «Готово». У результаті у полі `private` класу `CRGRDlg` створиться змінна `m_nData` типу `int`, а у методі `void CRGRDlg::DoDataExchange(CDataExchange* pDX)` з'явиться запис `DDX_Text(pDX, IDC_EDIT1, m_nData)` та у конструктор класу `CRGRDlg` буде додано поле ініціалізації `m_nData(0)`.

10. Вибрати елемент керування типу «List Box» та відкрити його властивості. Ознайомитися з його властивостями. Клікнути правою кнопкою мишки на елемент керування типу «List Box» та вибрати пункт меню «Добавить переменную». В результаті має відкритися вікно, що зображене на рис. 11. Задати значення полів так, як зображено на рис. 11. На даному рисунку вказується, що елемент керування з ідентифікатором `IDC_LIST1` зв'язується з об'єктом `m_DlgList` класу `CListCtrl` який буде розташований у полі `private` класу `CRGRDlg`. Натиснути «Готово». У результаті у полі `private` класу `CRGRDlg` створиться об'єкт `m_DlgList` класу `CListCtrl`, а у методі `void CRGRDlg::DoDataExchange(CDataExchange* pDX)` з'явиться запис `DDX_Control(pDX, IDC_LIST1, m_DlgList)`.

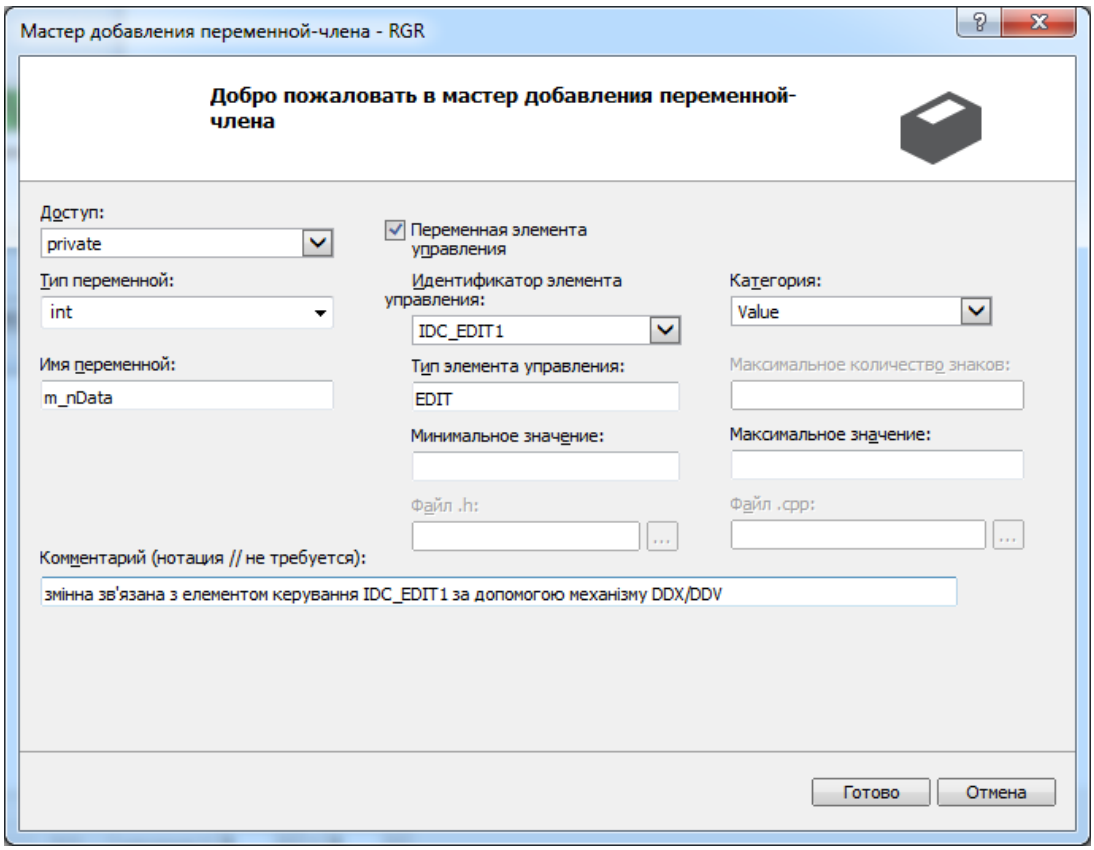

Рис. 10. Діалог зв'язування змінної з елементом керування типу «Edit Control»

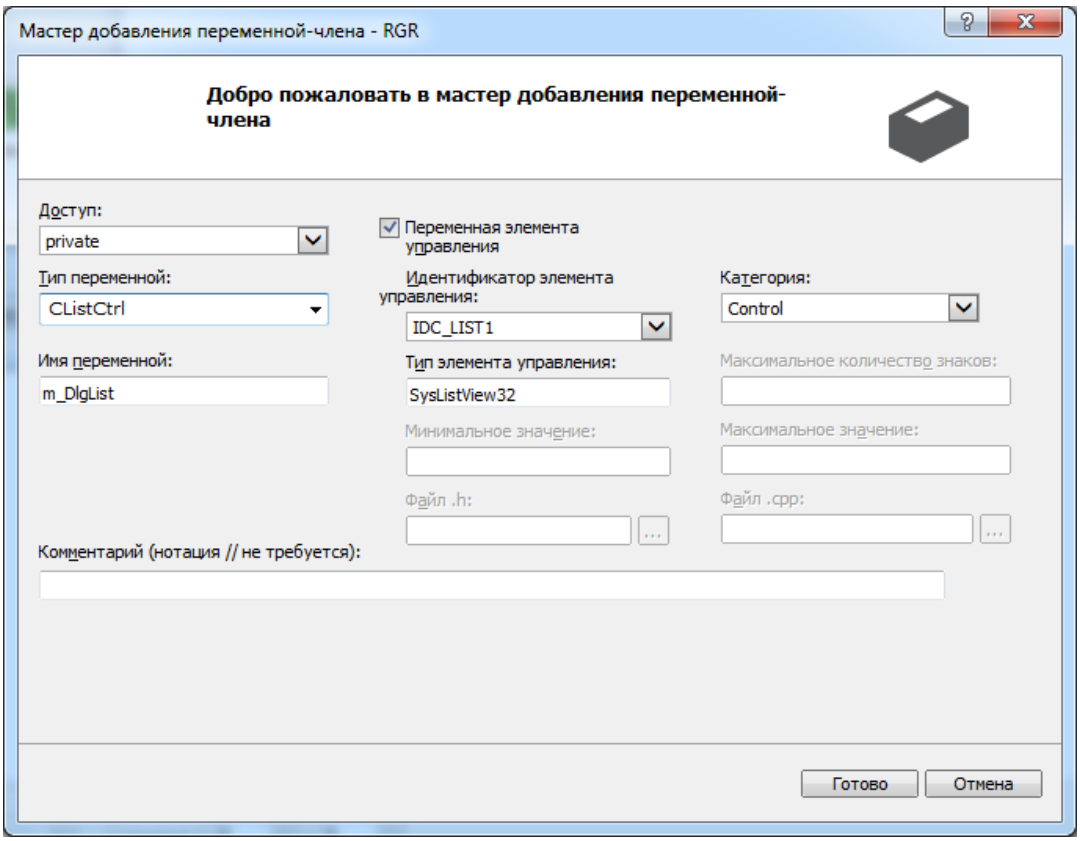

Рис. 11. Діалог зв'язування об'єкту з елементом керування типу «List Box»

11. Вибрати елемент керування типу «Button» з текстом «Button1» та відкрити його властивості. Ознайомитися з його властивостями. У полі «Подпись» переназвати кнопку на «Записати дані». Кнопку «Button2» переназвати на «Прочитати дані». Клікнути правою кнопкою мишки на кнопку «Записати дані» та вибрати пункт меню «Добавить обработчик событий…». У вікні, що відкриється, встановити значення так, як зображено на рис. 12 та натиснути «Добавить/править». На рис. 12 вказується, що у клас `CRGRDlg` буде додано обробник події (метод) натискання на кнопку (BN_CLICKED), який називатиметься `OnWriteData`. Те саме повторити для кнопки «Прочитати дані», назвавши обробник події `OnReadData`. В результаті цих дій у клас `CRGRDlg` будуть додані 2 методи – `OnWriteData` і `OnReadData`, в макрос `BEGIN_MESSAGE_MAP(CRGRDlg, CDialogEx)` будуть додані 2 макроси – `ON_BN_CLICKED(IDC_BUTTON1,&CRGRDlg::OnWriteData)` і `ON_BN_CLICKED (IDC_BUTTON2, &CRGRDlg::OnReadData)`, які вказують диспетчеру повідомлень, що повідомлення про натискання кнопки з ідентифікатором `IDC_BUTTON1` слід обробляти методом `CRGRDlg::OnWriteData`. Аналогічно для другого макроса.

12. Відкрити тіло методу обробника події натискання на кнопку «Записати дані» і добавити в нього код для буферизованого запису даних у файл:

```
void CRGRDlg::OnWriteData()
{
	// Записуємо значення з контролів в змінні
	UpdateData(true);

	// перевіряємо чи буфер не повний
	if (lst.size()<3)
		lst.insert(lst.end(), m_nData);
	else
```

```
	{

		CFileDialog OpenDialog(false, "txt", NULL,
			OFN_HIDEREADONLY,
			"Text Files (*.txt)|*.txt||");

		if(OpenDialog.DoModal() == IDOK) // Якщо натиснули кнопку OK в діалозі
		{
			char buf[10];

			// Відкрити файл за заданим шляхом в режимі запису
			try
			{
				CStdioFile file(OpenDialog.GetPathName(), CFile::modeCreate |
					CFile::modeNoTruncate | CFile::modeWrite);

				// переміщуємося в кінець файлу
				file.SeekToEnd();


				// записуємо дані з буфера (списку) в текстовий файл
				list<int>::iterator i;
				for(i=lst.begin(); i != lst.end(); i++)
				{
					sprintf_s(buf,"%d\n",*i);
					file.WriteString(buf);
				}

				// записуємо у файл останню порцію даних, яка не потрапила в буфер
				sprintf_s(buf,"%d\n",m_nData);
				file.WriteString(buf);

				// Закриваємо файл
				file.Close();

				// очистити буфер
				lst.clear();
			}
			catch (CInvalidArgException* ex)
			{
				MessageBoxA("Cannot open file");
				return;
			}
			catch (CFileException* ex)
			{
				MessageBoxA("Error writing file");
				return;
			}
			catch (...)
			{
				MessageBoxA("Unknown error");
				return;
			}
		}
	}
}
```

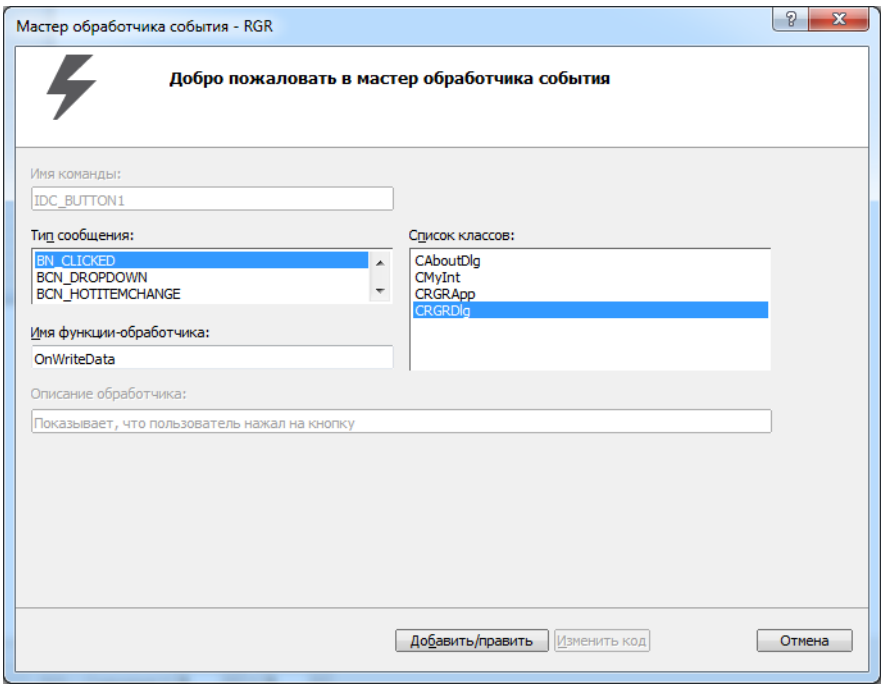

Рис. 12. Діалог створення обробника події натискання на кнопку

13. Налаштувати стилі відображення «List Box». Для цього у тілі методу `BOOL CRGRDlg::OnInitDialog()` після коментаря `// TODO: добавьте дополнительную инициализацию` додаємо код для програмного налаштування «List Box»:

```
// TODO: добавьте дополнительную инициализацию

// Встановлюємо стиль списку "Рапорт"
m_DlgList.ModifyStyle(LVS_LIST, LVS_REPORT);
// Визначаємо розмір клієнтської області списку
CRect rect;
m_DlgList.GetClientRect(&rect);
// Встановлюємо в список одну колонку, яка називається Data,
// розмір якої рівний ширині клієнтської області списку (ширині списку).
// Елементи вирівнюватимуться по лівій границі
m_DlgList.InsertColumn(0,_T("Data"),LVCFMT_LEFT,rect.Width());
```

Зауваження. Метод `OnInitDialog()` – це останній метод, що викликається перед відображенням діалогу на екрані монітора, тому всі налаштування елементів керування прийнято робити тут, оскільки вони вже створені, але ще не відображені.

14. Відкрити тіло методу обробника події натискання на кнопку «Прочитати дані» і добавити в нього код для читання даних з файлу і відображення їх у контролі типу «List Box»:

```
void CRGRDlg::OnReadData()
{

	CString str;
	int i=0;

	// Створюємо діалог вибору файлу для читання
	CFileDialog OpenDialog(true, "txt", NULL,
		OFN_FILEMUSTEXIST |
		OFN_HIDEREADONLY,
		"Text Files (*.txt)|*.txt||");
```

```
        // Якщо натиснули кнопку OK в діалозі
        if(OpenDialog.DoModal() == IDOK)
        {
                // Стираємо весь вміст контрола
                m_DlgList.DeleteAllItems();

                // Відкриваємо файл за заданим шляхом в режимі читання
                CStdioFile file(OpenDialog.GetPathName(), CFile::modeRead);
                // переміщуємося на початок файлу
                file.SeekToBegin();

                // Читаємо по рядково файл до кінця
                while(file.ReadString(str))
                {
                        m_DlgList.InsertItem(i++,str.GetBuffer(10));
                }

                // Закриваємо файл
                file.Close();

                // переносимо вміст об'єктів в контроли
                UpdateData(false);
        }
}
```

Зауваження. У даному методі відсутній контроль коректності читання з файлу. При виконанні індивідуального завдання цей контроль має бути реалізований!

15. Вибрати діалог в ресурсах. Правою кнопкою мишки клікнути по діалогу та вибрати у випадаючому меню пункт «Свойства». В результаті має відкритися вікно з властивостями діалогу. У властивостях діалогу перейти на вкладку «Сообщения». На цій вкладці найти повідомлення `WM_CLOSE` та клікнути на ньому. В результаті має з'явитися стрілочка Combobox-а. Клікнути на неї і вибрати <Add> OnClose (див. рис. 13). В результаті цих дій створиться обробник (метод) повідомлення `WM_CLOSE` для діалогу і в макрос `BEGIN_MESSAGE_MAP(CRGRDlg, CDialogEx)` додається макрос `ON_WM_CLOSE()`.

    Зауваження. Повідомлення `WM_CLOSE` надсилається вікну, коли вікно має бути закритим. Наприклад, при кліку мишкою на хрестику закривання вікна.

16. Відкрити обробник повідомлення `WM_CLOSE` для діалогу в файлі з реалізаціями методів класу діалогу. Додати код, по запису вмісту буфери при виході з програми, щоб введені, але не записані дані записати у файл:

```
void CRGRDlg::OnClose()
{
        // Записуємо значення з контролів в змінні
        UpdateData(true);

        // Якщо буфер порожній то виходимо
        if (!lst.empty())
        {
                // Створюємо діалог вибору файлу для запису
                CFileDialog OpenDialog(false, "txt", NULL,
                        OFN_HIDEREADONLY,
                        "Text Files (*.txt)|*.txt||");

                // Якщо натиснули кнопку OK в діалозі
                if(OpenDialog.DoModal() == IDOK)
                {
```

```
                        char buf[10];

                        // Відкрити файл за заданим шляхом в режимі запису
                        CStdioFile file(OpenDialog.GetPathName(), CFile::modeCreate |
                                CFile::modeNoTruncate | CFile::modeWrite);
                        // переміщуємося в кінець файлу
                        file.SeekToEnd();
                        // записуємо дані з буфера (списку) в текстовий файл
                        list<int>::iterator i;
                        for(i=lst.begin(); i != lst.end(); i++)
                        {
                                sprintf_s(buf,"%d\n",*i);
                                file.WriteString(buf);
                        }

                        // Закриваємо файл
                        file.Close();

                        // очистити буфер
                        lst.clear();
                }
        }
                // виклик базового методу OnClose() для здійснення типових дій
                // по закриттю програми
                CDialogEx::OnClose();
}
```

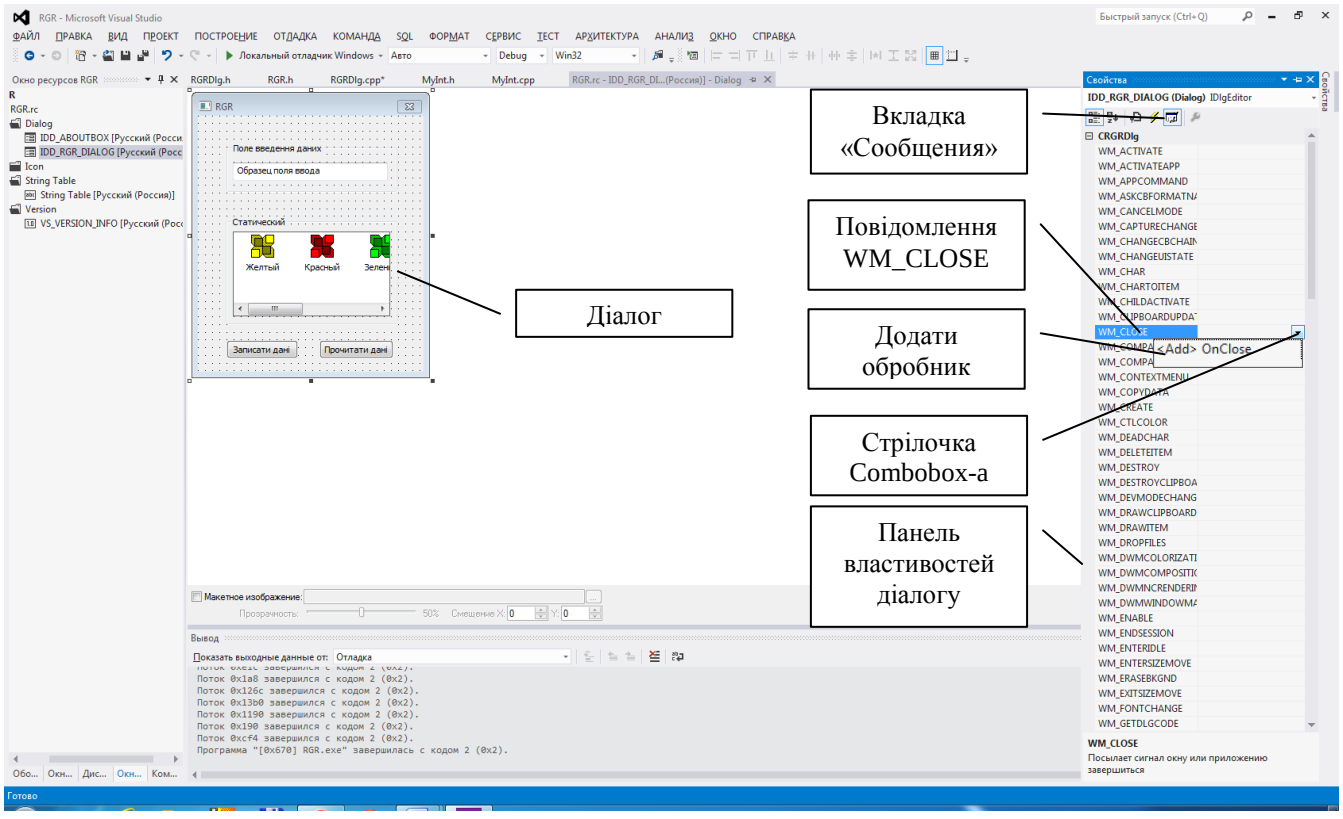

Рис.13. Створення обробника повідомлення `WM_CLOSE` для діалогу.

17. Відкрити вкладку ресурсів. Правою кнопкою мишки клікнути будь-де у вкладці ресурсів. Вибрати у виринаючому меню «Добавить ресурс…». У вікні, що відкриється вибрати «Menu» (див. рис. 14) і натиснути «Создать».

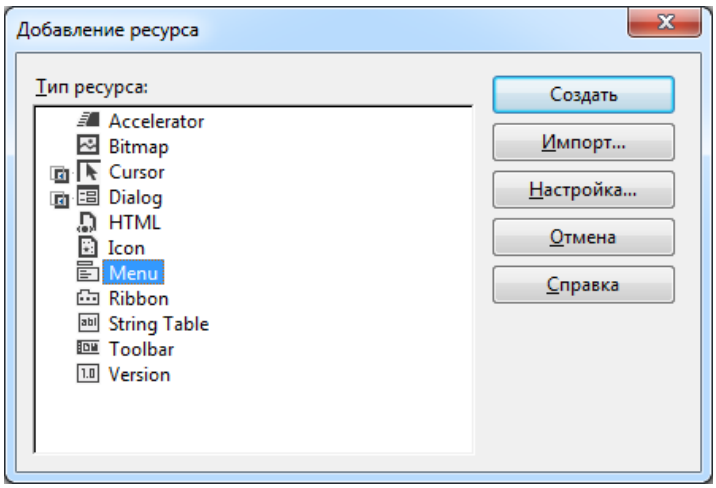

Рис. 14. Перелік ресурсів, які можна додати до діалогу

18. Ресурс «Меню» зображений на рис. 15. Для додання назв пунктів і підпунктів меню слід стати на порожню комірку пункту/підпункту меню мишкою і ввести назву пункту чи підпункту меню. Пункт меню має підпункти, якщо він має властивість «Всплывающее меню» («POP-UP»). Розділювачі вставляються властивістю «Разделитель» («Separator»). Для визначення «гарячої клавіші» для пунктів/підпунктів меню слід перед буквою, яка буде «гарячою клавішею», поставити символ ‘&’.

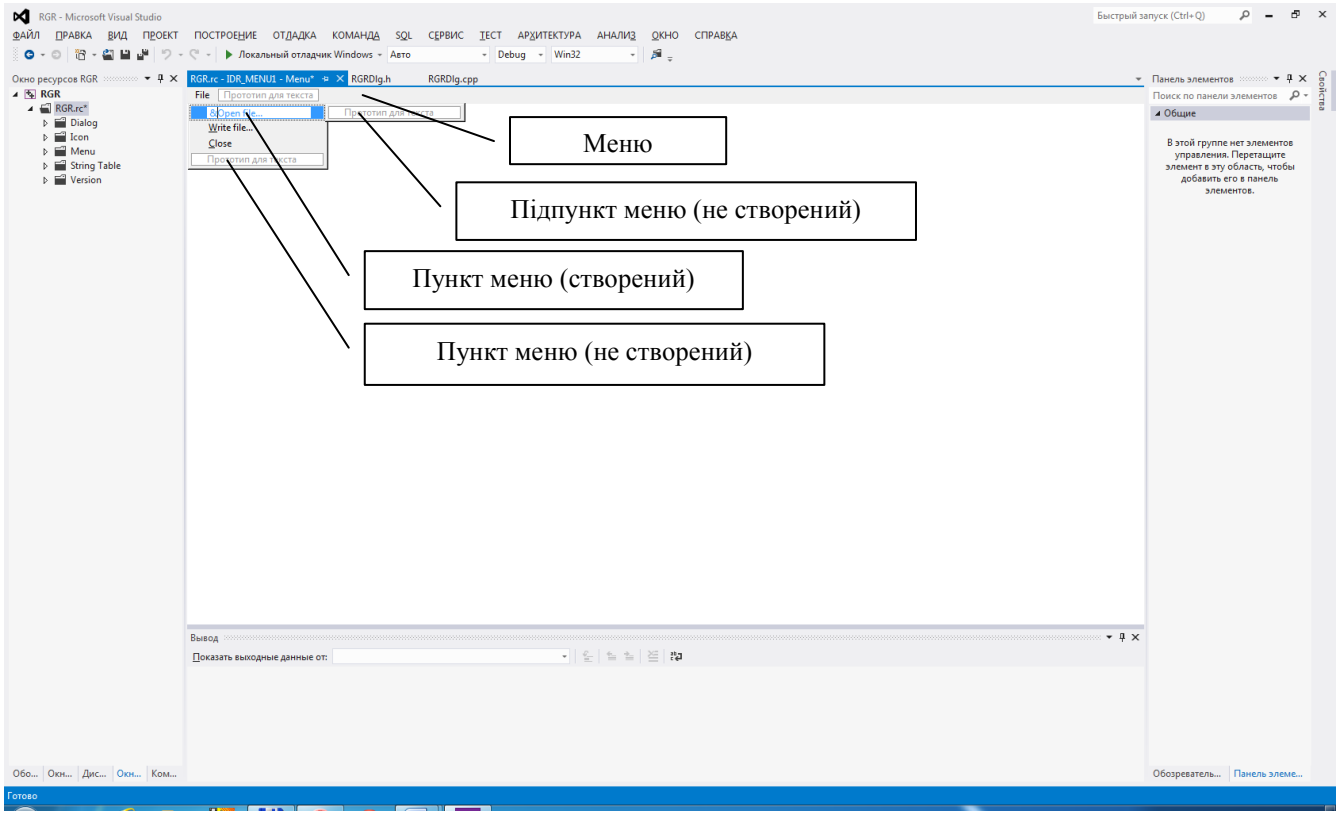

Рис. 18. Створення пунктів і підпунктів меню.

19. Створити обробники події для створених пунктів меню. Для цього слід клікнути правою кнопкою мишки на відповідному пункті меню, який не має властивості «Всплывающее меню». Вибрати «Добавить обработчик событий…» («Add Event Handler…») У вікні, що відкриється слід вибрати тип повідомлення `COMMAND` і клас діалогу в якому його слід створити обробник події (не `CAboutDlg`!!!), вибрати назву обробника, натиснути «Добавить/править» («Add and edit») (див. рис. 19). Відредагувати тіло обробника:

```
void CRGRDlg::OnFileFileRead()
{
	OnReadData();
}
```

Щоб меню і інші елементи програми мали однакові обробники подій слід зайти в макрос `BEGIN_MESSAGE_MAP(CRGRDlg, CDialogEx)` та відредагувати прив'язку події натискання на кнопку меню таким чином, щоб обробник події натискання на пункт меню і, наприклад, на кнопку мали однакові обробники:

`ON_COMMAND(ID_FILE_WRITEFILE, &CRGRDlg::OnWriteData)`, де `ID_FILE_WRITEFILE` – ідентифікатор пункту меню, `&CRGRDlg:: OnWriteData` – адреса обробника події натискання на пункт меню (в даному випадку співпадає з адресою обробника натискання на кнопку «Записати дані» (`ON_BN_CLICKED(IDC_BUTTON1, &CRGRDlg::OnWriteData)`)).

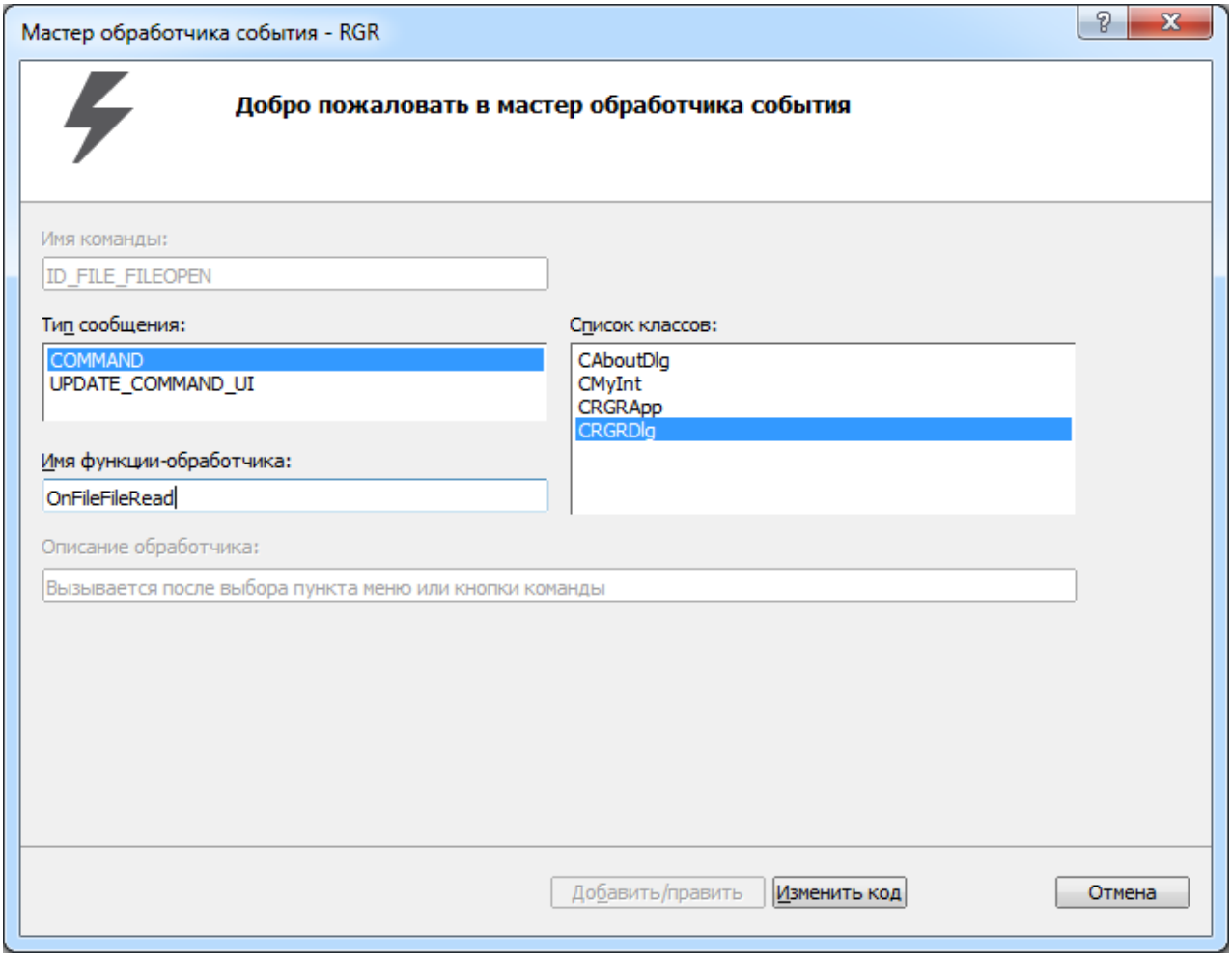

Рис.19. Створення обробника події натискання на пункт меню.

20. Повторити кроки 18-19 для кожного елементу меню.

21. Зв'язати меню з діалогом. Для цього слід вибрати у властивостях діалогу пункт «Свойства». У властивостях діалогу клацнути мишкою по пункту «Меню», після чого вибрати зі списку доступних меню ідентифікатор того меню, яке слід зв'язати з діалогом (IDR_MENU1).

22. Виконати індивідуальне завдання згідно варіанту, оформити і захистити звіт.

ПРИКЛАД ПРОГРАМИ

// ---------- **MyInt.h** ----------

```
#pragma once
class CMyInt
{
	int * m_pnNum;
public:
	/*
	 *  При використанні динамічного виділення пам'яті у членів-даних класу
	 *  слід обов'язково перевизначити конструктор за замовчуванням, конструктор копіювання,
	 *  деструктор, оператор присвоювання, які мають забезпечити коректне опрацювання
	 *  динамічно створених членів класу
	 */
	CMyInt(void);
	CMyInt(const CMyInt&);
	~CMyInt(void);

	CMyInt& operator=(const CMyInt&);

	void SetInt(int);
	int GetInt(void) const;

};
```

// ---------- **MyInt.cpp** ----------

```
#include "stdafx.h"
#include "MyInt.h"


CMyInt::CMyInt(void)
{
	m_pnNum = new int;
	*m_pnNum = 0;
}

CMyInt::CMyInt(const CMyInt& obj)
{
	m_pnNum = new int;
	*m_pnNum = *obj.m_pnNum;
}

CMyInt::~CMyInt(void)
{
	delete m_pnNum;
}

CMyInt& CMyInt::operator=(const CMyInt& obj)
{
	if (this != &obj)
	{
		*m_pnNum = *obj.m_pnNum;
		return *this;
	}
	return *this;
}
```

```
void CMyInt::SetInt(int n)
{
	*m_pnNum = n;
}

int CMyInt::GetInt(void) const
{
	return *m_pnNum;
}
```

// ---------- **Resource.h** ----------

```
//{{NO_DEPENDENCIES}}
// Включаемый файл, созданный в Microsoft Visual C++.
// Используется RGR.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_RGR_DIALOG                  102
#define IDR_MAINFRAME                   128
#define IDR_MENU1                       129
#define IDC_EDIT1                       1000
#define IDC_LIST1                       1001
#define IDC_BUTTON1                     1002
#define IDC_BUTTON2                     1003
#define ID_FILE_FILEREAD                32771
#define ID_FILE_WRITEFILE               32772
#define ID_FILE_FILECLOSE               32773

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        130
#define _APS_NEXT_COMMAND_VALUE         32776
#define _APS_NEXT_CONTROL_VALUE         1005
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

// ---------- **RGR.h** ----------

```
// RGR.h : главный файл заголовка для приложения PROJECT_NAME
//

#pragma once

#ifndef __AFXWIN_H__
	#error "включить stdafx.h до включения этого файла в PCH"
#endif

#include "resource.h"		// основные символы


// CRGRApp:
// О реализации данного класса см. RGR.cpp
//
```

```
class CRGRApp : public CWinApp
{
public:
	CRGRApp();

// Переопределение
public:
	virtual BOOL InitInstance();

// Реализация

	DECLARE_MESSAGE_MAP()
};

extern CRGRApp theApp;
```

// ---------- **RGRDlg.h** ----------

```
// RGRDlg.h : файл заголовка
//

#pragma once
#include "MyInt.h"
#include <list>
#include <iterator>
#include "afxcmn.h"
using namespace std;

// диалоговое окно CRGRDlg
class CRGRDlg : public CDialogEx
{
private:
	CMyInt m_intObj;
	list<int> lst;
// Создание
public:
	CRGRDlg(CWnd* pParent = NULL);	// стандартный конструктор

// Данные диалогового окна
	enum { IDD = IDD_RGR_DIALOG };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// поддержка DDX/DDV


// Реализация
protected:
	HICON m_hIcon;

	// Созданные функции схемы сообщений
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
private:
	int m_nData;
	CListCtrl m_DlgList;
public:
	afx_msg void OnWriteData();
	afx_msg void OnReadData();
	afx_msg void OnClose();
	afx_msg void OnFileFileRead();
	afx_msg void OnFileClose();
};
```

// ------------------------------------ **stdafx.h** ------------------------------------

```
// stdafx.h: включите файл для добавления стандартных системных файлов
//или конкретных файлов проектов, часто используемых,
// но редко изменяемых

#pragma once

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN            // Исключите редко используемые компоненты из заголовков
Windows
#endif

#include "targetver.h"

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // некоторые конструкторы CString будут явными

// отключает функцию скрытия некоторых общих и часто пропускаемых предупреждений MFC
#define _AFX_ALL_WARNINGS

#include <afxwin.h>         // основные и стандартные компоненты MFC
#include <afxext.h>         // расширения MFC


#include <afxdisp.h>        // классы автоматизации MFC


#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>           // поддержка MFC для типовых элементов управления Internet
Explorer 4
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // поддержка MFC для типовых элементов управления Windows
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxcontrolbars.h>     // поддержка MFC для лент и панелей управления


#ifdef _UNICODE
#if defined _M_IX86
#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-
Controls' version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#elif defined _M_X64
#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-
Controls' version='6.0.0.0' processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#else
#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-
Controls' version='6.0.0.0' processorArchitecture='*' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#endif
#endif
```

// ------------------------------------**targetver.h** ------------------------------------

```
#pragma once
// Включение SDKDDKVer.h обеспечивает определение самой последней доступной платформы Windows.

// Если требуется выполнить построение приложения для предыдущей версии Windows, включите
WinSDKVer.h и
// задайте для макроопределения _WIN32_WINNT значение поддерживаемой платформы перед
вхождением SDKDDKVer.h.
#include <SDKDDKVer.h>
```

// ---------- **RGR.cpp** ----------

```
// RGR.cpp : Определяет поведение классов для приложения.
//

#include "stdafx.h"
#include "RGR.h"
#include "RGRDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CRGRApp

BEGIN_MESSAGE_MAP(CRGRApp, CWinApp)
	ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()


// создание CRGRApp

CRGRApp::CRGRApp()
{
	// поддержка диспетчера перезагрузки
	m_dwRestartManagerSupportFlags = AFX_RESTART_MANAGER_SUPPORT_RESTART;

	// TODO: добавьте код создания,
	// Размещает весь важный код инициализации в InitInstance
}


// Единственный объект CRGRApp

CRGRApp theApp;


// инициализация CRGRApp

BOOL CRGRApp::InitInstance()
{
	// InitCommonControlsEx() требуется для Windows XP, если манифест
	// приложения использует ComCtl32.dll версии 6 или более поздней версии для включения
	// стилей отображения. В противном случае будет возникать сбой при создании любого окна.
	INITCOMMONCONTROLSEX InitCtrls;
	InitCtrls.dwSize = sizeof(InitCtrls);
	// Выберите этот параметр для включения всех общих классов управления, которые
	// необходимо использовать
	// в вашем приложении.
	InitCtrls.dwICC = ICC_WIN95_CLASSES;
	InitCommonControlsEx(&InitCtrls);

	CWinApp::InitInstance();


	AfxEnableControlContainer();

	// Создать диспетчер оболочки, в случае, если диалоговое окно содержит
	// представление дерева оболочки или какие-либо его элементы управления.
	CShellManager *pShellManager = new CShellManager;

	// Активация визуального диспетчера "Классический Windows" для включения
	// элементов управления MFC
```

```
	CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerWindows));

	// Стандартная инициализация
	// Если эти возможности не используются и необходимо уменьшить размер
	// конечного исполняемого файла, необходимо удалить из следующих
	// конкретных процедур инициализации, которые не требуются
	// Измените раздел реестра, в котором хранятся параметры
	// TODO: следует изменить эту строку на что-нибудь подходящее,
	// например на название организации
	SetRegistryKey(_T("Локальные приложения, созданные с помощью мастера приложений"));

	CRGRDlg dlg;
	m_pMainWnd = &dlg;
	INT_PTR nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Введите код для обработки закрытия диалогового окна
		//  с помощью кнопки "OK"
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Введите код для обработки закрытия диалогового окна
		//  с помощью кнопки "Отмена"
	}
	else if (nResponse == -1)
	{
		TRACE(traceAppMsg, 0, "Предупреждение. Не удалось создать диалоговое окно,
поэтому работа приложения неожиданно завершена.\n");
		TRACE(traceAppMsg, 0, "Предупреждение. При использовании элементов управления MFC
для диалогового окна невозможно #define _AFX_NO_MFC_CONTROLS_IN_DIALOGS.\n");
	}

	// Удалить диспетчер оболочки, созданный выше.
	if (pShellManager != NULL)
	{
		delete pShellManager;
	}

	// Поскольку диалоговое окно закрыто, возвратите значение FALSE, чтобы можно было выйти
	// из приложения вместо запуска генератора сообщений приложения.
	return FALSE;
}
```

// ---------- RGRDlg.cpp ----------

```
// RGRDlg.cpp : файл реализации
//

#include "stdafx.h"
#include "RGR.h"
#include "RGRDlg.h"
#include "afxdialogex.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// Диалоговое окно CAboutDlg используется для описания сведений о приложении

class CAboutDlg : public CDialogEx
{
```

```
public:
	CAboutDlg();

	// Данные диалогового окна
	enum { IDD = IDD_ABOUTBOX };

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // поддержка DDX/DDV

	// Реализация
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()


// диалоговое окно CRGRDlg


CRGRDlg::CRGRDlg(CWnd* pParent /*=NULL*/)
	: CDialogEx(CRGRDlg::IDD, pParent)
	, m_nData(0)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CRGRDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_EDIT1, m_nData);
	DDX_Control(pDX, IDC_LIST1, m_DlgList);
}

BEGIN_MESSAGE_MAP(CRGRDlg, CDialogEx)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, &CRGRDlg::OnWriteData)
	ON_BN_CLICKED(IDC_BUTTON2, &CRGRDlg::OnReadData)
	ON_WM_CLOSE()
	ON_COMMAND(ID_FILE_FILEREAD, &CRGRDlg::OnFileFileRead)
	ON_COMMAND(ID_FILE_WRITEFILE, &CRGRDlg::OnWriteData)
	ON_COMMAND(ID_FILE_FILECLOSE, &CRGRDlg::OnFileClose)
END_MESSAGE_MAP()


// обработчики сообщений CRGRDlg

BOOL CRGRDlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();
```

```
	// Добавление пункта "О программе..." в системное меню.

	// IDM_ABOUTBOX должен быть в пределах системной команды.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		BOOL bNameValid;
		CString strAboutMenu;
		bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
		ASSERT(bNameValid);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Задает значок для этого диалогового окна. Среда делает это автоматически,
	//  если главное окно приложения не является диалоговым
	SetIcon(m_hIcon, TRUE);			// Крупный значок
	SetIcon(m_hIcon, FALSE);		// Мелкий значок

	// TODO: добавьте дополнительную инициализацию

	// Встановлюємо стиль списку "Рапорт"
	m_DlgList.ModifyStyle(LVS_LIST, LVS_REPORT);
	// Визначаємо розмір клієнтської області списку
	CRect rect;
	m_DlgList.GetClientRect(&rect);
	// Встановлюємо в список одну колонку, яка називається Data,
	// розмір якої рівний ширині клієнтської області списку (ширині списку).
	// Елементи вирівнюватимуться по лівій границі
	m_DlgList.InsertColumn(0,_T("Data"),LVCFMT_LEFT,rect.Width());

	return TRUE;  // возврат значения TRUE, если фокус не передан элементу управления
}

void CRGRDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialogEx::OnSysCommand(nID, lParam);
	}
}

// При добавлении кнопки свертывания в диалоговое окно нужно воспользоваться приведенным ниже
// кодом,  чтобы нарисовать значок. Для приложений MFC, использующих модель документов или
// представлений, это автоматически выполняется рабочей областью.

void CRGRDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // контекст устройства для рисования

		SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
```

```
		// Выравнивание значка по центру клиентского прямоугольника
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Нарисуйте значок
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialogEx::OnPaint();
	}
}

// Система вызывает эту функцию для получения отображения курсора при перемещении
//  свернутого окна.
HCURSOR CRGRDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}


void CRGRDlg::OnWriteData()
{
	// Записуємо значення з контролів в змінні
	UpdateData(true);

	// перевіряємо чи буфер не повний
	if (lst.size()<3)
		lst.insert(lst.end(), m_nData);
	else
	{

		CFileDialog OpenDialog(false, "txt", NULL,
			OFN_HIDEREADONLY,
			"Text Files (*.txt)|*.txt||");

		if(OpenDialog.DoModal() == IDOK) // Якщо натиснули кнопку OK в діалозі
		{
			char buf[10];

			// Відкрити файл за заданим шляхом в режимі запису
			try
			{
				CStdioFile file(OpenDialog.GetPathName(), CFile::modeCreate |
					CFile::modeNoTruncate | CFile::modeWrite);

				// переміщуємося в кінець файлу
				file.SeekToEnd();
				// записуємо дані з буфера (списку) в текстовий файл
				list<int>::iterator i;
				for(i=lst.begin(); i != lst.end(); i++)
				{
					sprintf_s(buf,"%d\n",*i);
					file.WriteString(buf);
				}

				// записуємо у файл останню порцію даних, яка не потрапила в буфер
				sprintf_s(buf,"%d\n",m_nData);
				file.WriteString(buf);
```

```
                        // Закриваємо файл
                        file.Close();

                        // очистити буфер
                        lst.clear();
                }
                catch (CInvalidArgException* ex)
                {
                        MessageBoxA("Cannot open file");
                        return;
                }
                catch (CFileException* ex)
                {
                        MessageBoxA("Error writing file");
                        return;
                }
                catch (...)
                {
                        MessageBoxA("Unknown error");
                        return;
                }
        }

    }
}


void CRGRDlg::OnReadData()
{

    CString str;
    int i=0;

    // Створюємо діалог вибору файлу для читання
    CFileDialog OpenDialog(true, "txt", NULL,
            OFN_FILEMUSTEXIST |
            OFN_HIDEREADONLY,
            "Text Files (*.txt)|*.txt||");

    // Якщо натиснули кнопку OK в діалозі
    if(OpenDialog.DoModal() == IDOK)
    {
            // Стираємо весь вміст контрола
            m_DlgList.DeleteAllItems();

            // Відкриваємо файл за заданим шляхом в режимі читання
            CStdioFile file(OpenDialog.GetPathName(), CFile::modeRead);
            // переміщуємося на початок файлу
            file.SeekToBegin();



            // Читаємо по рядково файл до кінця
            while(file.ReadString(str))
            {
                    m_DlgList.InsertItem(i++,str.GetBuffer(10));
            }

            // Закриваємо файл
            file.Close();

            // переносимо вмість об'єктів в контроли
            UpdateData(false);
    }
}
```

```
void CRGRDlg::OnClose()
{
	// Записуємо значення з контролів в змінні
	UpdateData(true);

	// Якщо буфер порожній то виходимо
	if (!lst.empty())
	{
		// Створюємо діалог вибору файлу для запису
		CFileDialog OpenDialog(false, "txt", NULL,
			OFN_HIDEREADONLY,
			"Text Files (*.txt)|*.txt||");

		if(OpenDialog.DoModal() == IDOK) // Якщо натиснули кнопку OK в діалозі
		{
			char buf[10];

			// Відкрити файл за заданим шляхом в режимі запису
			CStdioFile file(OpenDialog.GetPathName(), CFile::modeCreate |
					CFile::modeNoTruncate | CFile::modeWrite);
			// переміщуємося в кінець файлу
			file.SeekToEnd();
			// записуємо дані з буфера (списку) в текстовий файл
			list<int>::iterator i;
			for(i=lst.begin(); i != lst.end(); i++)
			{
				sprintf_s(buf,"%d\n",*i);
				file.WriteString(buf);
			}

			// Закриваємо файл
			file.Close();

			// очистити буфер
			lst.clear();
		}
	}
	// виклик базового методу OnClose() для здійснення типових дій по закриттю програми
	CDialogEx::OnClose();

}


void CRGRDlg::OnFileFileRead()
{
	OnReadData();
}


void CRGRDlg::OnFileClose()
{
	SendMessageA(WM_CLOSE);

}
```

// ---------- **stdafx.cpp** ----------

```
// stdafx.cpp: исходный файл, содержащий стандарт, включающий
// RGR.pch будет предварительно откомпилированным заголовком
// stdafx.obj будет содержать предварительно откомпилированные сведения о типе

#include "stdafx.h"
```

НАВЧАЛЬНЕ ВИДАННЯ

ПРОГРАМУВАННЯ, ЧАСТИНА 2 (ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ)

МЕТОДИЧНІ ВКАЗІВКИ

до виконання розрахунково-графічної роботи для студентів базового напряму "Комп'ютерна інженерія"

Укладачі Морозов Юрій Васильович
Олексів Максим Васильович

Редактор

Комп'ютерне верстання

Здано у видавництво . Підписано до друку
Формат 70x100/16. Папір офсетний. Друк на різографі
Умовн. друк. арк. Обл..-вид. арк..
Тираж прим. Зам..

Видавництво Національного університету "Львівська політехніка"
Реєстраційне свідоцтво ДК №751 від 27.12.2001 р.

Поліграфічний центр Видавництва
Національного університету "Львівська політехніка"

Вул. Ф. Колесси, 2. Львів, 79000