ЛЕКЦІЯ 11.
СТРУКТУРА ПРОГРАМИ НА ОСНОВІ КЛАСІВ MFC

1. Файли, що включаються в проект. Домовленість, щодо імен в MFC.
2. Функція WinMain() та клас CWinApp.
3. Обробка повідомлень в MFC.

Файли, що включаються в проект

Середовище Visual C++ містить діалоговий інструмент, так званий “майстер”, для інтерактивного створення проекту. Цей інструмент дозволяє створити заготовки проектів усіх основних типів програм, що використовуються в Windows. При створенні проекту генерується заготовка основної програми та деякі допоміжні файли, що необхідні для компіляції проекту вибраного типу. Розглянемо основні файли, що автоматично включаються в проект віконної програми з використанням MFC.

Основним файлом, що включається у віконну програму є ‘StdAfx.h’. Він служить для включення в програму описання всіх основних бібліотечних функції, що необхідні для даного типу програми. Включення цього файлу в проект дозволяє ефективно використовувати механізм попередньої компіляції, що значно економить час на повторну компіляцію проекту.

В свою чергу файл ‘StdAfx.h’ включає заголовкові файли:

AFX.H – включає класи загального призначення, базові типи MFC і заголовки базових функцій C;

AFXRES.H – включає стандартні ідентифікатори ресурсів;

WINDOWS.H – включає описання функцій, структур та макровизначень Win32 API та для зручності розбитий на чотири частини WINDDEF.H, WINBASE.H, WINGDI.H та WINUSER.H;

WINDDEF.H – містить визначення базових типів;

WINBASE.H – містить описання функцій Win32 API;

WINGDI.H та WINUSER.H – містить оголошення функцій модулів GDI та USER і відповідні їм типи, структури та макровизначення.

Домовленість, що імен в MFC

Для іменування класів у MFC прийнято вживати префікс ‘C’ (наприклад CwinApp, CDialog, CWnd).

Іменування функцій (методів) є контекстно зрозумілим. Для використовуються три способи:

- об’єднують дієслово та іменник (наприклад, LoadIcan – завантажити іконку, DrawText – намалювати текст);
- структурована назва складається тільки з іменника (наприклад, DialogBox – блок діалогу);
- вказується напрямок перетворення даних (наприклад, XtoY – з X в Y).

Для змінних членів класу MFC використовується префікс ‘m_’ (від class members) за яким іде префікс, що характеризує тип даних на завершення структуроване ім’я змінної (наприклад, ‘m_pMainWnd’, де p – вказує на тип вказівник).

Функція WinMain() та клас CWinApp

При створенні проекту за допомогою “майстра” проектів, функція WinMain() явно не створюється. Вона захована в типовий об’єктний код заготовки програми. Такий підхід звільняє розробника від написання багатьох рядків рутинного додаткового коду для виконання стандартних кроків ініціалізації програми. Основні моменти в реалізації цієї функції розглядалися в попередніх лекціях. Тут слід зауважити, що у файлі реалізації основного класу програми CWinApp, створюється єдиний глобальний об’єкт цього класу з фіксованим іменем theApp, який і служить точкою входу в програму, а оболонка використовує це наперед визначене ім’я для запуску програми. В бібліотеці реалізовано функція, яка повертає вказівник на цей глобальний об’єкт.

CWinApp* pApp = AfxGetApp();

Клас CWinApp

Клас CWinApp є базовим класом від якого утворюється обов’язковий об’єкт-програма для Windows на основі MFC, його ієрархія показана на рис. __.

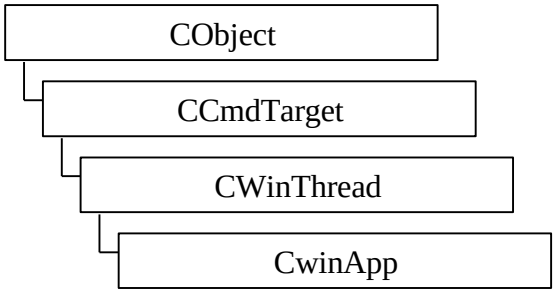

Рис. Ієрархія класу CWinApp

Основними задачами цього класу є:

- ініціалізація основного вікна програми;
- створення та опитування черги повідомлень;
- ініціалізація основних змінних, відновлення та збереження налаштувань і оточення програми.

Розглянемо основні змінні- та функції-члени класу CWinApp.

Змінні-члени класу:

LPSTR m_pszAppName – назва програми (загальнодоступна).

HINSTANCE m_hInstance – копія логічного номеру (дескриптора) програми (загальнодоступна).

LPSTR m_lpCmdLine – копія вказівника на командний рядок програми (загальнодоступна).

int m_nCmdShow – копія параметру, що визначає первинне відображення вікна програми на екрані (загальнодоступна).

LPSTR m_pszExeName – назва завантажувального модуля програми без розширення (загальнодоступна).

LPSTR m_pszProfileName – назва INI-файлу програми (загальнодоступна).

LPCSTR m_pszRegistryKey – визначає повний ключ реєстру, де зберігається профіль налаштувань програми (загальнодоступна).

Додатково клас CWinApp на слідує від класу CWinThread змінну m_pMainWnd типу CWnd, яка зберігає вказівник на об’єкт основного вікна біжучого потоку. Windows автоматично завершує програму, якщо закривається основне вікно.

Функції ініціалізації програми:

void LoadStdProfileSetting(UINT nMaxNRU = _AFX_MRU_COUNT) – завантажує список останніх файлів, що використовувалися (_AFX_MRU_COUNT=4).

void SetRegistryKey(LPCSTR lpszRegistryKey) – заставляє програму зберігати первинні на лаштування в реєстрі Windows а не в INI-файлі. Ключ lpszRegistryKey, за яким зберігаються ініціалізаційні дані програми в реєстрі має формат:

HKEY_CURENT_USER\Software\<НазваОрганізації>\<Програма>\<Секція>\<Змінна>

void EnableShellOpen() – дозволяє запускати програму подвійним клацанням миші на файлі, що асоціюється з програмою.

void RegistryShellFileTypes() – реєструє в реєстрі Windows типи файлів, що асоціюється з програмою.

void ParseCommandLine(CCommandLineInfo& rCmdInfo) – здійснює синтаксичний та семантичний аналіз командного рядка, що передається програмі та поміщає результати в об’єкт rCmdInfo.

BOOL ProcessShellCommand(CCommandLineInfo& rCmdInfo) – обробляє дані (аргументи) командного рядка.

UINT GetProfileInt(LPCTSTR lpszSection, LPCTSTR lpszEntry, int Def) – при успішному пошуку - повертає ціле число, що міститься полі lpszEntry секції lpszSection реєстру чи INI-файлу, інакше значення за замовчуванням (Def).

BOOL WriteProfileInt(LPCTSTR lpszSection,LPCTSTR lpszEntry,int nVal) – записує ціле число (nVal) у відповідне поле секції реєстру чи INI-файлу. Якщо ключова запис відсутня то вона створюється.

CString GetProfileString(…), BOOL WriteProfileString(…) – читання, запис символьного рядка.

BOOL GetProfileBinary(…), BOOL WriteProfileBinary(…) – читання, запис двійкового масиву даних.

Основні віртуальні функції керування програмою:

virtual BOOL InitInstance() – викликається кожен раз при ініціалізації нового (додаткового) екземпляру програми. Її наповнення повністю визначається розробником (в базовому класі, вона порожня). У ній зчитуються ініціалізаційні дані, реєструється та створюється основне вікно програми та інше.

virtual BOOL ExitInstance() – викликається з методу Run() для завершення біжучого екземпляру програми. У ній реалізовується збереження налаштувань програми в реєстрі чи INI-файлі, звільняються виділені програмі ресурси (ресурси стандартних елементів створених MFC звільняється автоматично).

virtual BOOL Run() – запускає та обслуговує основний цикл повідомлень програми, доки не отримає повідомлення WM_QUIT. Якщо черга повідомлень порожня, то запускає віртуальну функцію OnIdle() для фонової обробки.

virtual BOOL OnIdle(LONG lCount) – викликається в циклі обробки повідомлень, коли черга повідомлень порожня. Вона обновлює команди користувацького інтерфейсу, перемальовує елементи керування та інше. При пере визначення цієї функції обов’язково необхідно викликати функцію CwinnApp::OnIdle() базового класу. lCount – лічильник вкладених викликів.

Реєстрація класу вікна та його створення

Реєстрація та створення основного вікна програми на основі класів MFC майже нічим не відрізняється від подібної процедури на основі Win32 API, що розглядалася раніше. Основні відмінності спостерігаються при використання архітектури “Документ/вигляд”, яка буде розглянута на наступній лекції.

Обробка повідомлень в MFC

Основний цикл обробки повідомлень реалізований в методі Run() класу CwinnApp. Він діє аналогічно до розглянутого раніше циклу обробки повідомлень Win32 API за винятком запуску методу OnIdle() для виконання фонових операцій коли черга повідомлень порожня.

Категорії повідомлень MFC. У бібліотеці MFC прийнята класифікація повідомлень відмінна від Win32 API. Усі повідомлення розділено на три категорії:

- повідомлення Windows;
- повідомлення елементів керування;
- командні повідомлення (команди).

Цей розподіл зумовлений, що повідомлення кожної з категорій обробляються по-різному. В першу категорію входять повідомлення імена яких починаються з префіксу WM_, за винятком WM_COMMAND. Ці повідомлення обробляються вікнами і мають відповідні параметри які визначають алгоритм обробки. Сюди входять і апаратні повідомлення.

Друга категорія *повідомлення* (ивещения – notification message) від елементів керування та інших дочірніх вікон, що направляються батьківським вікнам. Наприклад, елемент керування посилає повідомлення своєму батьківському вікну, коли необхідно обновити інформацію про елементи списку. Батьківське вікно заповнює відповідну структуру та відсилає її назад елементу керування.

Механізми передачі та обробки повідомлень перших двох категорій повідомлень в основному однакові. Вони призначені для обробки віконними процедурами.

Третя категорія охоплює всі повідомлення WM_COMMAND, що називаються командами (або командними повідомленнями), від інтерфейсу користувача до якого входять меню, кнопки панелей інструментів и акселератори. Обробка команд відрізняється від обробки інших повідомлень і може проводитися багатьма об’єктами (документами, шаблонами документів і самим об’єктом “програма”).

Але незалежно від категорії повідомлення їх обробка здійснюється спеціальними методами якого-небудь класу, що мають спеціальну назву *обробник повідомлень* та призначені для обробки тільки одного повідомлення.

Карта повідомлень. Для того, щоб об’єкт міг обробляти повідомлення необхідно привести у відповідність повідомлення та обробник, що відповідає за його опрацювання. Для цього служить карта повідомлень. Карта повідомлень реалізує механізм пересилання повідомлень та команд Windows у вікна, документи та інші об’єкти реалізовані на базі MFC. Карти повідомлень перетворюють повідомлення у виклик відповідних функцій класів, що їх обробляють. Кожен клас, який може отримати повідомлення, повинен мати свою карту повідомлень. Вона повинна визначатися за межами будь-якого С-блоку.

Для визначення карти повідомлень в MFC використовують три макроси: BEGIN_MESSAGE_MAP, END_MESSAGE_MAP та DECLARE_MESSAGE_MAP.

Макрос DECLARE_MESSAGE_MAP повинен розміщуватися в кінці оголошення класу, що використовує карту повідомлень.

Структура карти повідомлень є набором макросів, що взяті в спеціальні “операторні дужки” (формуються автоматично при використання майстрів створення класу в середовищі Visual C++):

```
BEGIN_MESSAGE_MAP(CTheClass, CBaseClass)
  //{{AFX_MSG_MAP(CTheClass)
  ON_WM_CREATE()
  ON_WM_DESTROY()
  ON_COMMAND(ID_CHAR_BOLD, OnCharBold)
  ON_UPDATE_COMMAND_UI(ID_CHAR_BOLD, OnUpdateCharBold)
  //}}AFX_MSG_MAP
  ON_NOTIFY(FM_GETFORMAT, ID_VIEW_FORMATBAR,
OnGetCharFormat)
  ON_NOTIFY(FM_SETFORMAT, ID_VIEW_FORMATBAR, OnSetCharFormat)
END_MESSAGE_MAP()
```

Параметрами макросу BEGIN_MESSAGE_MAP() є ім’я класу для якого описується карта повідомлень (CTheClass) та ім’я його батьківського класу (CBaseClass). Між викликами цих двох макросів розміщені *компоненти карти повідомлень*.

Компоненти карти повідомлень. Для стандартного повідомлення Windows визначений свій макрос у формі:
ON_WM_XXX // XXX – ім’я повідомлення, наприклад ON_WM_CREATE

Імена обробників повідомлень формуються на основі наступної угоди: ім’я завжди починається з префіксу On, за яким йде ім’я відповідного повідомлення Windows без префіксу WM_. Наприклад для повідомлення WM_CREATE в класі CWnd визначено обробник:
afx_msg void OnCreate();

Описання всіх стандартних обробників повідомлень Windows можна знайти в файлі AFXWIN.H.

Командні повідомлення від меню, акселераторів та кнопок панелей інструментів обробляються макросом
ON_COMMAND(id, memberFunc)

Параметрами цього макросу є: id – ідентифікатор команди та memberFunc – довільне ім’я (але бажано дотримуватися угод Windows при іменування обробників) функції обробника повідомлення.

Для команд оновлення елементів керування використовується той же механізм але інший макрос
ON_UPDATE_COMMAND_UI(id, memberFunc)

А обробники таких команд мають вигляд.
afx_msg void memberFunc(CCmdUI *pCmdUI);

В обох випадках обробник memberFunc буде викликано тільки тоді, коли у віконну процедуру поступить команда з ідентифікатором id у параметрі wParam.

Наступні команди дозволяють поставити у відповідність команди, що посилаються елементами керування та дочірніми вікнами своїм предкам та відповідні обробники. Для цього застосовується макрос
ON_CONTROL(wNotifyCode, id, memberFunc)

Цей макрос використовує як параметри код повідомлення wNotifyCode від елементу керування id та довільний обробник memberFunc. Обробник викликається тільки тоді, якщо код повідомлення елемента (наприклад BN_CLICKED) співпаде із зареєстрованим в карті повідомлень значенням wNotifyCode, а значення параметра співпаде з id.

Для найбільш поширених повідомлень від елементів керування використовуються спеціальні макроси. Наприклад:
ON_BN_CLICKED(id, memberFunc)
ON_EN_FOCUS(id, memberFunc)
ON_LBN_BDLCLK(id, memberFunc)
та інші. Повний список макросів цієї групи є в файлі AFXMSG.H.

Існують макроси для зв’язування одного обробника з декількома командами чи повідомленнями від елементів керування.
ON_COMMAND_RANGE(idFirst, idLast, memberFunc)
ON_CONTROL_RANGE(wNotifyCode, idFirst, idLast, memberFunc)
ON_NOTiFY_RANGE(wNotifyCode, idFirst, idLast, memberFunc)

ON_UPDATE_COMMAND_UI_RANGE(idFirst, idLast, memberFunc)

Команди, що визначаються користувачем включаються в карту повідомлень за допомогою макросу:

ON_MESSAGE(WM_NAMEMSG, OnNameMsg)

Параметрами якого є номер (WM_MESSAGE) та ім’я (OnNameMsg) обробника повідомлення. Для того щоб номер повідомлення був унікальним в межах системи (програми) застосовують наступне оголошення:

#define WM_NAMEMSG (WM_USER + numb)

де numb – довільне додатне число (комбінація повинна бути унікальною).

Повідомлення, що визначаються користувачем включаються в карту за допомогою макросу:

ON_REGISTERED_MESSAGE(nMessageVariables, memberFunc)

де nMessageVariables – зареєстроване в Windows повідомлення (для отримання унікального ідентифікатора необхідно викликати функцію RegisterWindowMessage()), a memberFunc його обробник.

Для команд та повідомлень користувача обробники мають наступний прототип:

afx_msg LRESULT memberFunc(WPARAM wParam, LPARAM lParam);

де через wParam та lParam у функцію передається додаткова до повідомлення інформація.

Стандартний маршрут команди. Команди обробляються за механізмом трохи відмінним від стандартного циклу повідомлень Windows. Оскільки команди формуються в результаті взаємодії користувача з програмою, то, зазвичай, команда починає свій шлях до отримувача від основного вікна програми.

Кожен об’єкт, що має карту повідомлень здатний обробляти команди. Якщо в карті повідомлень не знайдено відповідного обробника, то команда посилається батьківському об’єкту і так далі. Якщо відповідного обробника і далі не знайдено, то команда посилається процедурі обробки команд за замовчуванням.

Порядок в якому адресати пересилають команди зашитий в бібліотеці класів MFC і містить наступні кроки:

1. Команду отримує фрейм MDI (об’єкт класу CMDIFrameWnd). Порядок отримувачів:
   - активне вікно класу CMDIChildWnd;
   - сам об’єкт класу CMDIFrameWnd;
   - програма класу CWinApp.
2. Команду отримує фрейм документу (об’єкт класу CFrameWnd, CMDIChildWnd) Порядок отримувачів:
   - активний вид (view);
   - біжучий фрейм документу;
   - програма класу CWinApp.
3. Команду отримує вид (view). Порядок отримувачів:
   - біжучий вид;

- документу приєднаний до виду.

4. Команду отримує документ. Порядок отримувачів:
- біжучий документ;
- шаблон документу, приєднаний до нього.

5. Команду отримує блок діалогу. Порядок отримувачів:
- біжучий блок діалогу;
- вікно, що володіє цим блоком діалогу;
- програма класу CWinApp.

Стандартне повідомлення Windows не шукає відповідного обробника повідомлення воно посилається безпосередньо у віконну процедуру того об’єкту якому воно призначене.

Функції роботи з повідомленнями. Загальні функції роботи з повідомленнями, такі як SednMessage(), PosrMessage() розглядалися раніше їх призначення таке ж як і в відповідних системних функцій. Додатково розглянемо функцію реєстрації повідомлень користувача. Її прототип:

UINT ::RegisterWindowMessage(LPCTSTR lpString);

Повертає унікальний ідентифікатор (в діапазоні від 0xC000 до 0xFFF) нового повідомлення. Як параметр lpString вона використовує вказівник на символьний рядок, що визначає ім’я повідомлення. Зареєстроване повідомлення діє до кінця сеансу роботи з Windows.

Приклад реєстрації повідомлення користувача.

```
class CUserWnd : public CParentWnd {

        …

    //{{AFX_MSG_MAP(CUserWnd)
    afx_msg LRESULT OnSample(WPARAM wParam, LPARAM lParam);
    //}}AFX_MSG_MAP
  DECLARE_MESSAGE_MAP()
}

UINT WM_SAMPLE = RegisterWindowMessage(“MESSAGE_SAMPLE”);

BEGIN_MESSAGE_MAP(CUserWnd, CParentWnd)
  //{{AFX_MSG_MAP(CUserWnd)
  ON_REGISTERED_MESSAGE(WM_SAMPLE, OnSample)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```