Тема 6. Віконні програми Windows

План
1. Функція WinMain()
   1.1. Реєстрація класу вікна
   1.2. Створення вікна
   1.3. Цикл повідомлень
2. Процедура вікна (Функція обробки повідомлень)

Кожна Windows-програма з графічним інтерфейсом (віконна програма або GUI-програма) має два суттєвих компоненти: головну функцію **WinMain()**, яка ініціалізує вікно, і функцію **WindowProc()**, яка обслуговує повідомлення Windows. Ця структура становить основу всіх програм Windows.

Функція **WinMain()** викликається Windows при запуску кожної віконної програми. Задачами цієї функції є реєстрація класу вікна, створення примірника вікна заданого класу та відображення його на екрані. Вона також містить цикл повідомлень для одержання із черги повідомлень, адресованих програмі.

Функція **WindowProc()**, яка інколи має ім'я **WndProc()** або й інше, викликається операційною системою кожного разу, коли повідомлення має передаватися вікну програми. Така функція називається функцією із зворотним викликом (Callback Function). Зазвичай кожне вікно програми має власну функцію **WindowProc()**. Задачею цієї функції є обробка повідомлень, адресованих вікну програми.

1. Функція WinMain()

Функція **WinMain()** це точка входу до програми Windows. Вона ініціалізує програму, відображує вікно програми на екрані і запроваджує основний цикл повідомлень.

int WINAPI WinMain(HINSTANCE *hInstance*, **HINSTANCE** *hPrevInstance*, **LPSTR** *lpCmdLine*, **int** *nCmdShow* **);**

hInstance	дескриптор примірника. Це 32-розрядне число, що ідентифікує примірник програми в середовищі ОС. Цей номер надає Windows, коли програма запускається на виконання.
hPrevInstance	дескриптор попереднього примірника, завжди NULL. Це спадок від 16-розрядної Windows.
lpCmdLine	параметри командного рядка (програми Windows можуть бути запущені з командного рядка або іншим процесом).
NCmdShow	визначає спосіб відображення вікна (мінімізоване, максимізоване або приховане).

Функція **WinMain()** завершується, коли отримує повідомлення WM_QUIT. Повертає код завершення, що міститься в параметрі wParam.

Реєстрація класу вікна

Спочатку треба визначити, якого типу вікно потрібно створити. Цей тип вікна називають класом вікна (не плутати із класами C++).

Відомості про клас вікна містить спеціальна структура типу **WNDCLASSEX** (раніше **WNDCLASS**). Перед створенням вікна потрібно створити змінну типу WNDCLASSEX, і надати значення кожному з її елементів (подібно до заповнення бланка).

```
typedef struct _WNDCLASSEX {
    UINT    cbSize;          // розмір структури
    UINT    style;           // стиль класу
    WNDPROC lpfnWndProc;     // вказівник на віконну функцію
    int     cbClsExtra;      // додаткові байти класу
    int     cbWndExtra;      // додаткові байти примірника вікна
```

```
    HANDLE  hInstance;        // дескриптор примірника
    HICON   hIcon;            // дескриптор іконки
    HCURSOR hCursor;          // дескриптор курсору
    HBRUSH  hbrBackground;    // дескриптор пензля фону
    LPCTSTR lpszMenuName;     // вказівник на ім'я меню
    LPCTSTR lpszClassName;    // вказівник на ім'я класу
    HICON   hIconSm;          // мала іконка вікна
} WNDCLASSEX;
```

Створюємо змінну:

```
WNDCLASSEX wc;
```

Заповнюємо поля структури:

```
wc.cbSize         = sizeof(WNDCLASSEX);
wc.style          = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc    = WindowProc;
wc.cbClsExtra     = 0;                    // немає додаткових байтів після класу
wc.cbWndExtra     = 0;                    // вікна та після примірника вікна
wc.hInstance      = hInstance;            // дескриптор примірника програми
wc.hIcon          = LoadIcon(0, IDI_APPLICATION);     // іконка програми
wc.hCursor        = LoadCursor(0, IDC_ARROW);         // курсор програми
wc.hbrBackground  = GetStockObject(GRAY_BRUSH);       // сірий пензлик для фону
wc.lpszMenuName   = 0;                                // меню немає
wc.lpszClassName  = "MyWindClass";                    // ім'я класу
wc.hIconSm        = 0;                                // мала іконка
```

Після того, як заповнено усі змінні, їх можна передавати у Windows, щоб зареєструвати клас. За це відповідає функція `RegisterClassEx()`.

```
if(!RegisterClassEx(&wc))
{
  MessageBox(NULL, "Window Registration Failed!", "Error!",
      MB_ICONEXCLAMATION | MB_OK);
  return 0;
};
```

Для застарілої структури є функція реєстрації `RegisterClass()`.

Створення вікна

Після реєстрації класу вікна можна створювати примірники вікна, посилаючись на цей клас. Для цього використовують функцію **CreateWindow()**, яка повертає дескриптор створеного вікна. Цей дескриптор потрібно зберегти для подальшого звертання до вікна. Функція використовує дані із структури **WNDCLASSEX** через вказівник на ім'я зареєстрованого класу lpClassName.

HWND CreateWindow(

```
LPCTSTR lpClassName,        // вказівник на ім'я зареєстрованого класу
LPCTSTR lpWindowName,       // вказівник на ім'я вікна
DWORD dwStyle,              // стиль вікна
int x,                      // горизонтальна та
int y,                      // вертикальна позиція вікна
int nWidth,                 // ширина вікна
int nHeight,                // висота вікна
HWND hWndParent,            // дескриптор батьківського вікна (або власника)
HMENU hMenu,                // дескриптор меню або ідентифікатор дочірнього вікна
HANDLE hInstance,           // дескриптор примірника програми
LPVOID lpParam              // вказівник на дату створення вікна
  );
```

Наприклад:

```
HWND hWnd;                          // дескриптор вікна
```

```
...
hWnd = CreateWindow(
            szClassName,            // ім'я класу вікна
            "My First Window",      // заголовок вікна
            WS_OVERLAPPEDWINDOW,    // стиль вікна
            CW_USEDEFAULT,          // позиція лівого верхнього X
            CW_USEDEFAULT,          // кутка вікна на екрані      Y (за умовчанням)
            CW_USEDEFAULT,          // ширина за умовчанням
            CW_USEDEFAULT,          // висота за умовчанням
            0,                      // без батьківського вікна
            0,                      // без меню
            hInstance,              // дескриптор примірника програми
            0                       // без дати створення
            );
```

WS_OVERLAPPEDWINDOW вікно має рядок заголовка, границю, системне меню, кнопки *Згорнути*, *Розгорнути*, аналогічно комбінації прапорців WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX

Після виклику функції **CreateWindow()** вікно вже існує, але ще не відображується на екрані. Для відображення вікна слід викликати іншу функцію ShowWindow():

```
BOOL ShowWindow(hWnd, nCmdShow);                    // показати вікно
```

Тут є два параметри. Перший ідентифікує вікно і є дескриптором, поверненим функцією CreateWindow(). Другий – значення nCmdShow, який передавався WinMain() і вказує, як вікно з'являється на екрані.

Приклади: SW_HIDE, SW_RESTORE, SW_SHOWNORMAL

Після виклику функції **ShowWindow()**, вікно з'являється на екрані але все ще не має ніякого прикладного вмісту, отже потрібно перемалювати його клієнтську область. Цей код не може бути розташований у самій функції WinMain(), тому що клієнтська область перемальовується також у випадках, які не залежать від самої програми (наприклад, якщо поверх нашого вікна переміщують вікно іншої програми).

Windows діє таким чином: при необхідності перемалювати вікно, вона посилає системне повідомлення у **WindowProc()**, яка і відповідає за перемальовування вікна. Тому, якщо ми хочемо, щоб **WindowProc()** перемалювала вікно, потрібно попросити Windows послати системне повідомлення нашій **WindowProc()**. Для цього у **WinMain()** викликають функцію

```
BOOL UpdateWindow(HWND hWnd);
```

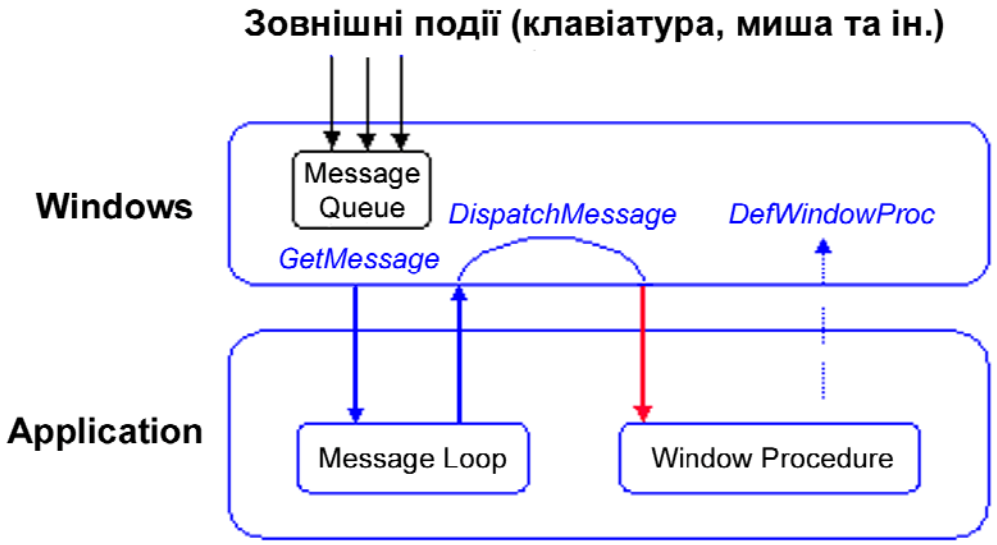
Якщо клієнтська область вікна не порожня, то функція посилає повідомлення WM_PAINT безпосередньо процедурі вказаного вікна, оминаючи чергу повідомлень програми.

Функції головної функції **WinMain()** на цьому не вичерпуються. Після відображення вікна ця функція займається обробкою повідомлень.

Цикл повідомлень

Коли під час роботи програми користувач переміщає мишу, набирає на клавіатурі, клацає по меню вікна, або подає будь-які інші команди, система генерує повідомлення, які надходять у чергу повідомлень програми.

Наступна задача функції **WinMain()** – діставати повідомлення із черги і реагувати на них відповідним чином. Коротко обробку повідомлень можна описати так: із черги дістається повідомлення, якщо це WM_QUIT, то програма завершується, якщо інше, то воно направляється процедурі вікна **WindowProc()** для обробки.



Обробка повідомлень здійснюється циклом:

```
while(GetMessage(&msg, 0, 0, 0) == TRUE)        // дістати повідомлення
{
      TranslateMessage(&msg);                    // транслювати повідомлення
      DispatchMessage(&msg);                     // відправити повідомлення
}
```

Функція **GetMessage()** дістає повідомлення із черги повідомлень програми і копіює його в структуру типу MSG. Вона повертає TRUE, доки не зустріне повідомлення WM_QUIT, тоді повертає FALSE і закінчує цикл повідомлень. Якщо немає жодного повідомлення, **GetMessage()** *блокується* системою, тобто чекає, поки не надійде повідомлення.

Функція **TranslateMessage()** переводить коди віртуальних клавіш на коди символів у повідомленні. Її слід використовувати, якщо програма повинна одержувати символьне введення із клавіатури.

Нарешті функція **DispatchMessage()** повертає повідомлення Windows, яка пересилає його вікну, якому воно було адресоване. Це може бути наше основне вікно або інше, як-то елемент керування або навіть інша програма. Про це не слід турбуватися, ми лише отримуємо повідомлення і посилаємо далі, система сама потурбується про те, щоб воно дісталося необхідного вікна.

Усі повідомлення, не оброблені процедурою нашого вікна, оброблюються процедурою за умовчанням **DefWindowProc()**.

2. Процедура вікна (Функція обробки повідомлень)

Функція **WinMain()** не містить нічого специфічного для нашої програми, окрім створення і відображення вікна. Весь код, який визначає поведінку програми, включається у ***процедуру обробки повідомлень***. Це функція **WindowProc()**, на яку посилається **WinMain()** через структуру **WNDCLASSEX**. Windows звертається до цієї функції кожного разу, коли надходить повідомлення для основного вікна програми.

Функція **WindowProc()**, аналізує потік повідомлень із черги, відбирає повідомлення, на які вона повинна відреагувати, і викликає відповідні функції (ділянки коду), відповідальні за обробку тих чи інших подій (обробники подій).

Функція WindowProc()

Прототип функції WindowProc():

```
LRESULT CALLBACK WindowProc(
 HWND hwnd,                    // дескриптор вікна
```

```
UINT uMsg,              // ідентифікатор повідомлення
WPARAM wParam,          // перший параметр повідомлення
LPARAM lParam           // другий параметр повідомлення
  );
```

Тип, що повертається, **LRESULT**, це тип Windows, який зазвичай еквівалентний **long**. Оскільки Windows звертається до функції через вказівник (у структурі WNDCLASSEX функції WinMain()), вона кваліфікується як **CALLBACK**.

Чотири параметри, які передаються, забезпечують інформацію про конкретне повідомлення. Значення кожного з цих параметрів описується в наступній таблиці:

Параметр	Значення
HWND hWnd	дескриптор вікна, в якому відбувається подія, що викликає повідомлення.
UINT uMsg	ідентифікатор повідомлення (ID), 32-розрядне значення, яке вказує тип повідомлення.
WPARAM wParam	32-розрядне значення, що містить додаткову інформацію залежно від типу повідомлення.
LPARAM lParam	32-розрядне значення, що містить додаткову інформацію залежно від типу повідомлення.

Вікно, якому адресоване вхідне повідомлення, ідентифікується першим параметром hWnd, що передається функції. Саме повідомлення ідентифікується значенням uMsg, яке передається WindowProc(). Це значення можна співставити із символічними константами, які описують повідомлення. Вони всі починаються з WM_ (Windows Message), типові приклади – WM_PAINT, яке відповідає запиту на перемальовування клієнтської зони вікна, WM_LBUTTONDOWN, яке вказує, що була натиснута ліва кнопка миші та ін.

Процес декодування повідомлення, яке посилає Windows, зазвичай виконується за допомогою оператора вибору **switch** у функції **WindowProc()** на підставі числового коду повідомлення. Вибір необхідного повідомлення здійснюється за допомогою оператора **case** для кожного типу повідомлення. Типова структура такої конструкції подана нижче:

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT msg,
                            WPARAM wParam, LPARAM lParam)
{
      switch(msg)
      {
      case WM_LBUTTONDOWN:
            MessageBox(hWnd, "Left Button Pressed", "Mouse Message", MB_OK);
            break;
      case WM_CLOSE:
            DestroyWindow(hwnd); break;
      case WM_DESTROY:
            PostQuitMessage(0); break;
      default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
      }
      return 0;
}
```

Кожна програма Windows має щось подібне у своєму коді, хоча це може бути прихованим від очей, наприклад, якщо ви пишете програми із використанням MFC або інших механізмів. У кожному випадку програма сприймає конкретне значення повідомлення і забезпечує відповідну обробку для цього повідомлення. Будь-які повідомлення, з якими програма не хоче мати справи, повинні бути повернуті назад у Windows, яка обробить їх за допомогою процедури, заданої за умовчанням. Повернення процедури у Windows здійснюється функцією **DefWindowProc()**.