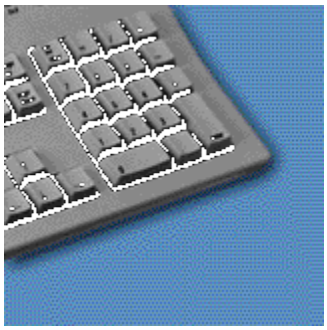


6. 메시지 맵(message map)

<장도비라>
이벤트를 메시지로 큐에 전송하고 메시지 큐를 처리하는 방식은 GUI 운영체제가 사용하는 일반적인 방식이다. 이러한 이벤트 중심적인 처리 방식은 일반 응용 프로그램을 만들 때도 적용할 수 있으며, 이러한 설계의 이점은 메시지를 처리하는 한 함수에서 모든 논리 처리를 한다는 것, 즉 이벤트에 집중하여 프로그램 할 수 있다는 것이다. 이러한 구조는 이벤트를 처리하는 코드 조각들이 소스의 여러 곳에 흩어지지 않도록 하므로, 객체 지향적이며, 발생할 수 있는 에러를 효과적으로 줄일 수 있다. 이 장에서 우리는 윈도우 메시지를 효과적으로 처리하기 위해 MFC가 사용한 방법, 즉 메시지 맵의 원리를 이해하기 위해 단계별로 메시지 맵을 작성한다! 또한 메시지 맵의 자동화를 위한 MFC의 매크로들을 살펴본다. 이 장의 내용은 10장에서 살펴볼 RTTI(run-time type identification)와 아울러 가장 중요한 내용이다.
</장도비라>

이 장에서 MFC의 **메시지 펌프(message pump)**□의 원리를 이해

<여기서 잠깐>
MFC는 메시지 루프의 동작을 메시지 펌프라 부른다. 메시지 펌프는 PeekMessage()와 DispatchMessage()의 기본 동작으로 구성된다.
</여기서 잠깐>

하기 위해 MFC의 메시지 맵의 원리를 직접 구현해 본다. 처음에는 순수한 C 프로그램으로 시작할 것이지만, 마지막 단계에서 매크로를 사용하는 완전한 메시지 맵을 구현할 것이다.

이 장에서 다음 사항들을 살펴 볼 것이다.

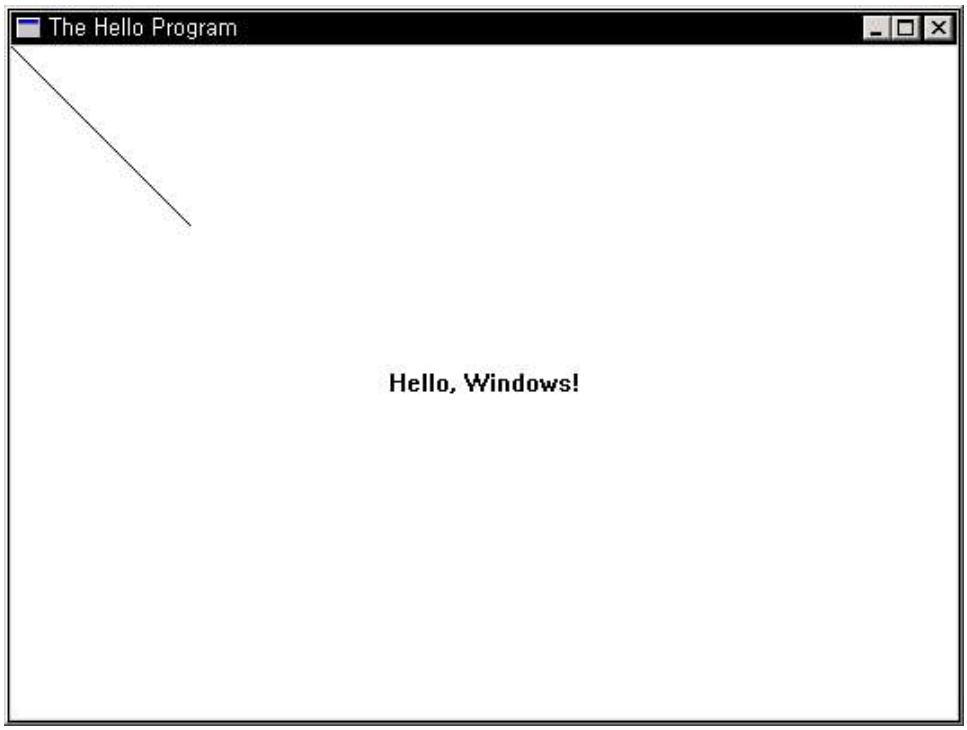
- MFC의 메시지 루프
- 메시지 맵의 원리: 단계적 구현

● 메시지 맵의 자동화를 위한 여러 매크로들

단계 0: 순수한 C

<절도비라>
이 절에서는 순수한 C 언어를 사용하여 윈도우 클라이언트 영역의 중앙에 Hello, Windows!를 출력하는 프로그램을 작성한다. 이 간단한 프로그램이 단계 5에서 마침내 메시지 맵을 사용하는 버전이 될 것이다!
</절도비라>

순수한 C만을 사용하여, 간단한 Hello 프로그램을 작성해 보자. 프로그램의 실행결과는 아래 [그림 6.1]과 같을 것이다.



[그림 6.1] Hello 프로그램: 클라이언트의 중앙에 Hello, Windows!를 출력한다.

작성할 첫 번째 프로그램은 화면의 중앙에 Hello, Windows!를 출력한다. 또한 마우스 왼쪽 버튼을 누르면 좌측 상단에 실선을 그린다.

또한 메시지 루프를 MFC의 것처럼 특별하게 만들 것인데, 메시지 큐에 메시지가 없으면 비주얼 C++의 디버거 창에 증가하는 숫자를 출력한다. 디버거 창의 모습은 아래 [그림 6.2]와 같다.

[그림 6.2] 디버거 창: 비주얼 C++의 디버거 창에는 증가하는 숫자를 출력한다.

윈도우 프로시저에서 마우스 왼쪽 버튼을 처리하는 부분은 다음과 같다.

```
...
    case WM_PAINT:
        hdc=BeginPaint(hwnd,&ps);
        //hdc = GetDC(hwnd);
        GetClientRect(hwnd,&rect);
        DrawText(hdc,"Hello, Windows!",-1,&rect,
            DT_SINGLELINE|DT_CENTER|DT_VCENTER);
        EndPaint(hwnd,&ps);
        //ReleaseDC(hwnd, hdc);
        return 0;
    case WM_LBUTTONDOWN:
        hdc = GetDC(hwnd);
        MoveToEx(hdc,0,0,NULL);
        LineTo(hdc,100,100);
        ReleaseDC(hwnd, hdc);
        return 0;
...
```

WM_PAINT에서는 BeginPaint()를 사용하고, 다른 곳에서는 GetDC()를 사용한 것에 주의하자. WM_LBUTTONDOWN에서 그래픽을 처리하는 방법은 좋지 않다. 윈도우가 무효화(invalidate)되었을 때, WM_LBUTTONDOWN 메시지는 발생하지 않을 것이므로, 이 부분은 화면에서 사라질 것이다.

MS-DOS 그래픽 프로그래밍의 경험이 있다면 그래픽을 모아서 처리하는 것은 이상하게 보일지도 모르지만, 그려할 할 부분을 한 곳에서 처리하는 것

은 일반적이며 당연한 것이다. 즉, 화면에 표시할 그래픽은 모두 WM_PAINT 에서 처리하도록 논리를 구성해야 한다.

우리는 1장에서 다음과 비슷한 메시지 루프를 만들었다.

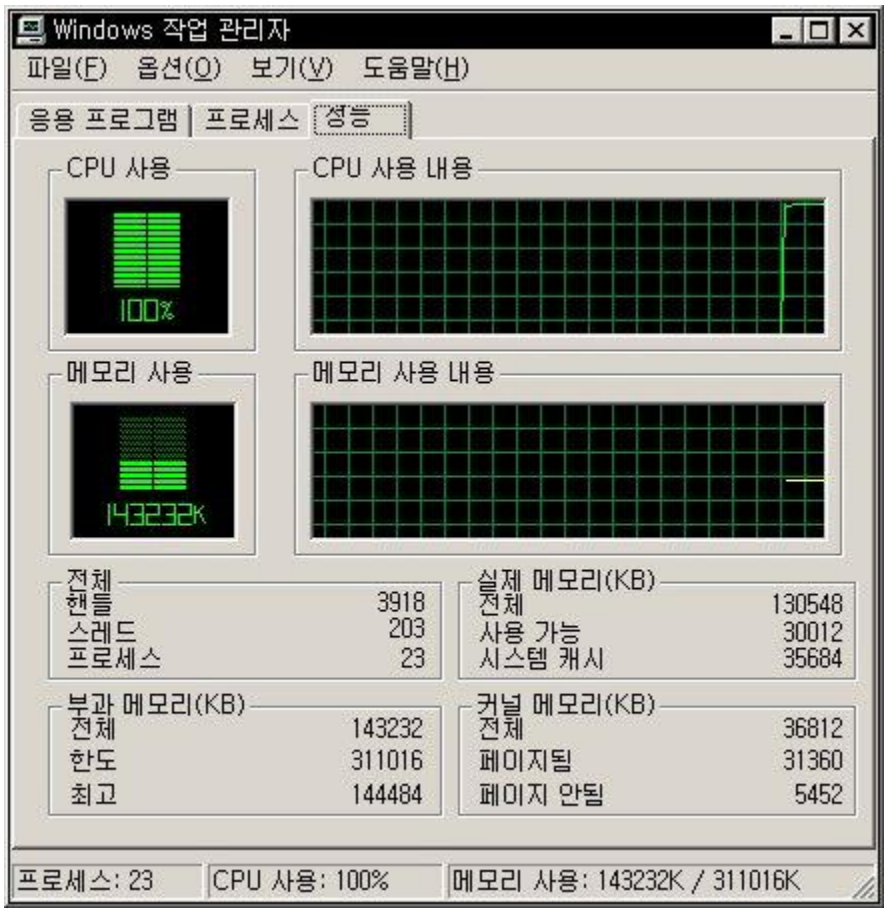
```
while (GetMessage(&msg,NULL,0,0))
{
    DispatchMessage(&msg);
}//while
```

사실 메시지 루프를 위와 같이 작성하는 것이 바람직하다. 위의 루프는 메시지가 있는 경우에만 윈도우 프로시저를 호출한다. 그러므로 처리할 메시지가 없다면 CPU는 블록 상태가 되어 다른 응용 프로그램이 CPU를 할당받아 일을 처리할 수 있는 것이다.

하지만, 이 예에서는 메시지 루프를 다음과 같이 작성했다.

```
while ( 1 )
{
    if ( PeekMessage( &msg, NULL, 0, 0, PM_REMOVE ) )
    {
        if ( msg.message == WM_QUIT ) break;
        DispatchMessage(&msg);
    }//if
    static int  i = 0;
    char        buffer[8];
    wsprintf( buffer, "%i\n", ++i );
    OutputDebugString( buffer );
}//while
```

위의 메시지 루프는 GetMessage() 대신에 PeekMessage()를 사용한다. 비슷해 보이지만, 메시지 루프를 위와 같이 작성하는 것은 바람직하지 않다. PeekMessage()는 블록 상태가 되지 않는다. GetMessage()는 메시지 큐에 메시지가 없을 경우, 블록 상태가 된다. 하지만, PeekMessage()는 메시지 큐에 메시지가 없는 경우 즉시 리턴된다. 그러므로 위의 메시지 루프는 while()문이 계속해서 실행되는 전형적인 비지 웨이팅(busy waiting)이다. 그러므로 위 프로그램을 실행하면 CPU를 거의 100%점유하는 프로그램이 될 것이다!



[그림 6.3] 비지 웨이팅 프로그램의 CPU 점유율: 비지 웨이팅을 하는 프로그램은 CPU를 거의 100% 점유하는 돼지(pig) 같은 프로그램이다.

MFC의 메시지 루프는 위와 비슷하지만 같지는 않다. 즉 사용자가 비지 웨이팅의 유무를 선택할 수 있는 유연한 구조를 가지고 있다. 메시지 루프에서 처리할 메시지가 없는 시간을 **아이들 타임(idle time)**이라 하자. MFC의 아이들 타임에 호출되는 가상 함수 OnIdle()을 제공한다. 사용자는 OnIdle()을 오버라이드하여 리턴값을 적절하게 선택하여 비지 웨이팅을 선택할 수 있다. 사용자가 OnIdle()에서 TRUE를 리턴하면 OnIdle()은 계속해서 호출된다. 후에 이 예를 작성해 볼 것이다.

비주얼 C++의 디버거 창에 정보를 출력하는 OutputDebugString()함수는 유용하다. 이 함수를 간단한 결과를 확인해 보는데 쓸 수 있을 뿐만 아니라, 특

정한 자동화를 처리하는데도 사용할 수 있다. 첫 번째 예의 전체 소스를 아래에 [예제 6.1]에 리스트하였다.

[예제 6.1] 단계 0 순수한 C

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam)
{
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    switch (iMsg)
    {
    case WM_CREATE:
        return 0;
    case WM_PAINT:
        hdc=BeginPaint(hwnd,&ps);
        //hdc = GetDC(hwnd);
        GetClientRect(hwnd,&rect);
        DrawText(hdc,"Hello, Windows!",-1,&rect,
            DT_SINGLELINE|DT_CENTER|DT_VCENTER);
        EndPaint(hwnd,&ps);
        //ReleaseDC(hwnd, hdc);
        return 0;
    case WM_LBUTTONDOWN:
        hdc = GetDC(hwnd);
        MoveToEx(hdc,0,0,NULL);
        LineTo(hdc,100,100);
        ReleaseDC(hwnd, hdc);
        return 0;
//    case WM_CLOSE:
//        hdc = GetDC(hwnd);
//        MoveToEx(hdc,0,0,NULL);
//        LineTo(hdc,100,100);
//        ReleaseDC(hwnd, hdc);
//        Sleep(2000);
        return 0;
    case WM_DESTROY:
        PostQuitMessage(0);
```

```
        return 0;
    }//switch
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc()

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow)
{
    static char szAppName[]="HelloWin";
    HWND        hwnd;
    MSG         msg;
    WNDCLASSEX  wndclass;

    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL,IDI_APPLICATION);

    RegisterClassEx(&wndclass);

    hwnd=CreateWindow(szAppName,    //window class name
        "The Hello Program",        //window caption
        WS_OVERLAPPEDWINDOW,        //window style
        CW_USEDEFAULT,              //initial x position
        CW_USEDEFAULT,              //initial y position
        CW_USEDEFAULT,              //initial x size
        CW_USEDEFAULT,              //initial y size
        NULL,                       //parent window handle
        NULL,                       //window menu handle
        hInstance,                  //program instance handle
        NULL);                      //creation parameters

    ShowWindow(hwnd,iCmdShow);
    //UpdateWindow(hwnd);
```

```
    /* //
    while (GetMessage(&msg,NULL,0,0))
    {
        //TranslateMessage(&msg);
        static int  i = 0;
        char        buffer[80];
        wsprintf( buffer, "%i\n", ++i );
        OutputDebugString( buffer );
        DispatchMessage(&msg);
    }//while
    /**/
    while ( 1 )
    {
        if ( PeekMessage( &msg, NULL, 0, 0, PM_REMOVE ) ) // (1)
        {
            if ( msg.message == WM_QUIT ) break;
            DispatchMessage(&msg);
        }//if
        static int  i = 0;
        char        buffer[8];
        wsprintf( buffer, "%i\n", ++i );
        OutputDebugString( buffer ); // (2)
    }//while
    return msg.wParam;
}//WinMain()
```

메시지 루프는 GetMessage()를 사용하지 않고, PeekMessage()를 사용한다(1). PeekMessage()는 메시지 큐에 메시지가 없는 경우 즉시 리턴되므로, 메시지가 없는 경우, 디버거 창에 계속해서 문자열을 출력한다(2).

단계 1: 클래스의 사용

<절도비라>
이 절에서는 앞 절의 단계 0에서 작성한 소스를 단순히 클래스를 사용하는 버전으로 바꿔보자. 이 장의 예를 통해 우리는 논리적인 단위들을 클래스의 멤버로 구성하는 법을 익힐 수 있을 것이다.
</절도비라>

단계 0의 소스를 클래스를 사용하는 버전으로 바꿀 수 있다. 개념 설명을 위해 MFC의 모든 기능으로 한꺼번에 바꾸지는 않고 차례대로 기능을 추가할 것이다. 먼저 단 한 개의 클래스만을 사용하는 버전을 작성해 보자.

WinMain()은 크게 다음과 같은 부분으로 구성된다.

1) 윈도우 클래스를 등록하고, 윈도우를 만드는 부분
2) 메시지 루프
3) 메시지 루프를 탈출한 뒤의 종료 부분

각각을 다음과 같이 구현할 수 있다.

1) 인스턴스를 초기화하는 함수 InitInstance()
2) 메시지 루프 Run()
3) 인스턴스를 탈출하는 ExitInstance()

그리고 처리하는 세 개의 메시지 각각을 담당하는 멤버 함수를 구현한다. 이것을 담당하는 클래스를 CApp라 하면 CApp는 다음과 같을 것이다.

```
class CApp {
    static char szAppName[];
    HWND        hwnd;
    MSG         msg;
    WNDCLASSEX  wndclass;

public:
    void InitInstance(HINSTANCE hInstance, PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();

    // message handler
    void OnCreate();
    void OnDraw();
    void OnDestroy();
};//class CApp
```

이제 WinMain()은 전역으로 선언된 CApp의 객체 app의 각각의 멤버 함수를 호출한다.

```
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow) {
    app.InitInstance(hInstance,szCmdLine,iCmdShow);
    app.Run();
    return app.ExitInstance();
}//WinMain
```

윈도우 프로시저도 각각의 메시지에 대응하는 app 객체의 멤버함수를 호출한다.

```
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam) {
    switch (iMsg) {
    case WM_CREATE:
        app.OnCreate();
        return 0;
    case WM_PAINT:
        app.OnDraw();
        return 0;
    case WM_DESTROY:
        app.OnDestroy();
        return 0;
    }//switch
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc
```

소스를 [예제 6.2]에 리스트하였다.

[예제 6.2] 단계 1 클래스의 사용

```
#include <windows.h>

//Forward declaration-------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam);

//Class definition----------------------------------------
class CApp {
    static char szAppName[];
    HWND        hwnd;
```

```
    MSG         msg;
    WNDCLASSEX  wndclass;

public:
    void InitInstance(HINSTANCE hInstance, PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();

    // message handler
    void OnCreate();
    void OnDraw();
    void OnDestroy();
};//class CApp

void CApp::InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                        int iCmdShow) {
    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL,IDI_APPLICATION);

    RegisterClassEx(&wndclass);

    hwnd=CreateWindow(szAppName,    //window class name
        "The Hello Program",        //window caption
        WS_OVERLAPPEDWINDOW,        //window style
        CW_USEDEFAULT,              //initial x position
        CW_USEDEFAULT,              //initial y position
        CW_USEDEFAULT,              //initial x size
        CW_USEDEFAULT,              //initial y size
        NULL,                       //parent window handle
        NULL,                       //window menu handle
        hInstance,                  //program instance handle
        NULL);                      //creation parameters
```

```
    ShowWindow(hwnd,iCmdShow);
    UpdateWindow(hwnd);
}//CApp::InitInstance

void CApp::Run() {
    while (GetMessage(&msg,NULL,0,0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }//while
}//CApp::Run

WPARAM CApp::ExitInstance() {
    return msg.wParam;
}//CApp::ExitInstance

char CApp::szAppName[]="HelloWin";

//Event handler-----------------------------------------
void CApp::OnCreate() {
}//CApp::OnCreate

void CApp::OnDraw() {
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    hdc=BeginPaint(hwnd,&ps);
    GetClientRect(hwnd,&rect);
    DrawText(hdc,"Hello, Windows!",-1,&rect,
        DT_SINGLELINE|DT_CENTER|DT_VCENTER);
    EndPaint(hwnd,&ps);
}//CApp::OnDraw

void CApp::OnDestroy() {
    PostQuitMessage(0);
}//CApp::OnDestroy

//Global object-----------------------------------------
CApp app;

//Window procedure--------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
```

```
                        LPARAM lParam) {
    switch (iMsg) {
    case WM_CREATE:
        app.OnCreate();
        return 0;
    case WM_PAINT:
        app.OnDraw();
        return 0;
    case WM_DESTROY:
        app.OnDestroy();
        return 0;
    }//switch
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow) {
    app.InitInstance(hInstance,szCmdLine,iCmdShow);
    app.Run();
    return app.ExitInstance();
}//WinMain
```

위의 구현은 자동화된 라이브러리에 적합하지 않다. 처리해야 할 메시지마다 멤버 함수를 구현해 주어야 하는 것은 당연하다. 하지만, 그 때마다 윈도우 프로시저를 수정하는 것은 바람직하지 않다. WinMain()과 윈도우 프로시저를 최종 구현자에게 완전히 숨기기를 원한다. 최종 구현자는 자신이 처리해야 할 메시지에만 집중해서 프로그램하는 것이다.

단계 2: 가상 함수의 이용

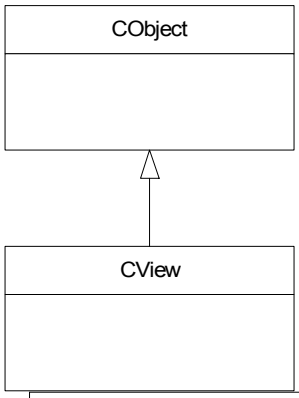
<절도비라>

이 절에서는 단계 1에서 처리하는 윈도우 메시지에 대응하는 가상함수를 구현한다. 또한, 단계 1에서 작성한 프로그램에 가상 함수를 사용하여 윈도우 프로시저와 WinMain()을 숨기는 기법을 이해한다.

</절도비라>

메시지에 상관없이 윈도우 프로시저를 변경시키지 않고 유지하는 한 가지

해결책은 모든 메시지에 대응하는 가상 함수를 미리 구현해 놓는 것이다. 또한, 위저드(wizard)에 의해서 결정될 클래스 이름에 종속적인 코드를 생성하지 않기 위해, 미리 구현된 베이스 클래스를 이용할 수 있다. 우리는 다음 [그림 6.4]와 같은 클래스 구조를 설계할 것이다.



[그림 6.4] 가상 함수의 이용: 공통적인 기능을 CObject에 미리 구현한다. 또한, 모든 메시지에 대응하는 가상함수를 CObject에 구현하여 두면, 사용자가 처리를 원하는 메시지에 대응하는 가상 함수만을 오버라이드하면 된다.

CObject는 다음과 같은 구조가 될 것이다.

```
class CObject {
protected:
    static char szAppName[];
    HWND        hwnd;
    MSG         msg;
    WNDCLASSEX  wndclass;

public:
    void InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();

    // message handler
    virtual void OnCreate()      = 0;
    virtual void OnDraw()        = 0;
    virtual void OnDestroy()     = 0;
    virtual void OnLButtonDown() = 0;
};//class CObject
```

이제 CObject를 상속받아 CView를 구현한다.

```
class CView : public CObject {
public:
    // override all message handler
    void OnCreate();
    void OnDraw();
    void OnDestroy();
    void OnLButtonDown();
};//class CView
```

이러한 구현은 WinMain()과 윈도우 프로시저를 자동화 할 수 있다. 우리는 미리 CObject 타입의 포인터를 하나 선언해 놓는다.

```
CObject* pCObject;
```

그리고, 사용자에 의해 지정된 클래스 이름으로 생성된 전역 객체의 시작 주소를 이 포인터에 대입하면, WinMain()과 윈도우 프로시저는 최종 사용자에게 숨겨질 수 있다. WinMain()과 윈도우 프로시저는 다음과 같은 모양이 될 것이다.

```
//Global object-----------------------------------------
CView app;

//Window procedure--------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam) {
    switch (iMsg) {
    case WM_CREATE:
        pCObject->OnCreate();
        return 0;
    case WM_PAINT:
        pCObject->OnDraw();
        return 0;
    case WM_DESTROY:
        pCObject->OnDestroy();
        return 0;
    case WM_LBUTTONDOWN:
        pCObject->OnLButtonDown();
```

```
        return 0;
    }//switch
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow) {
    pCObject = &app;
    pCObject->InitInstance(hInstance,szCmdLine,iCmdSh
    pCObject->Run();
    return pCObject->ExitInstance();
}//WinMain
```

소스를 [예제 6.3]에 리스트하였다.

[예제 6.3] 단계 2 가상함수의 이용

```
#include <windows.h>

//Forward declaration-------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam);

//Class CObject-------------------------------------------
class CObject {
protected:
    static char szAppName[];
    HWND        hwnd;
    MSG         msg;
    WNDCLASSEX  wndclass;

public:
    void InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();

    // message handler
    virtual void OnCreate()      = 0;
    virtual void OnDraw()        = 0;
    virtual void OnDestroy()     = 0;
    virtual void OnLButtonDown() = 0;
```

```
};//class CObject

void CObject::InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                           int iCmdShow) {
    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL,IDI_APPLICATION);

    RegisterClassEx(&wndclass);

    hwnd=CreateWindow(szAppName,     //window class name
        "The Hello Program",         //window caption
        WS_OVERLAPPEDWINDOW,         //window style
        CW_USEDEFAULT,               //initial x position
        CW_USEDEFAULT,               //initial y position
        CW_USEDEFAULT,               //initial x size
        CW_USEDEFAULT,               //initial y size
        NULL,                        //parent window handle
        NULL,                        //window menu handle
        hInstance,                   //program instance handle
        NULL);                       //creation parameters

    ShowWindow(hwnd,iCmdShow);
    UpdateWindow(hwnd);
}//CObject::InitInstance

void CObject::Run() {
    while (GetMessage(&msg,NULL,0,0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }//while
}//CObject::Run

WPARAM CObject::ExitInstance() {
```

```
    return msg.wParam;
}//CObject::ExitInstance

char CObject::szAppName[]="HelloWin";

//-----------------------------------------------
// pointer declaration for upcase(subtype principle)
CObject* pCObject;

//class CView-------------------------------------------
class CView : public CObject {
public:
    // override all message handler
    void OnCreate();
    void OnDraw();
    void OnDestroy();
    void OnLButtonDown();
};//class CView

//CView Event handler-----------------------------------------
void CView::OnCreate() {
}//CView::OnCreate

void CView::OnDraw() {
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    hdc=BeginPaint(hwnd,&ps);
    GetClientRect(hwnd,&rect);
    DrawText(hdc,"Hello, Windows!",-1,&rect,
        DT_SINGLELINE|DT_CENTER|DT_VCENTER);
    EndPaint(hwnd,&ps);
}//CView::OnDraw

void CView::OnDestroy() {
    PostQuitMessage(0);
}//CView::OnDestroy

void CView::OnLButtonDown() {
    MessageBox(NULL,"MESSAGE","TITLE",MB_ICONEXCLAMAT
}//CView::OnLButtonDown
```

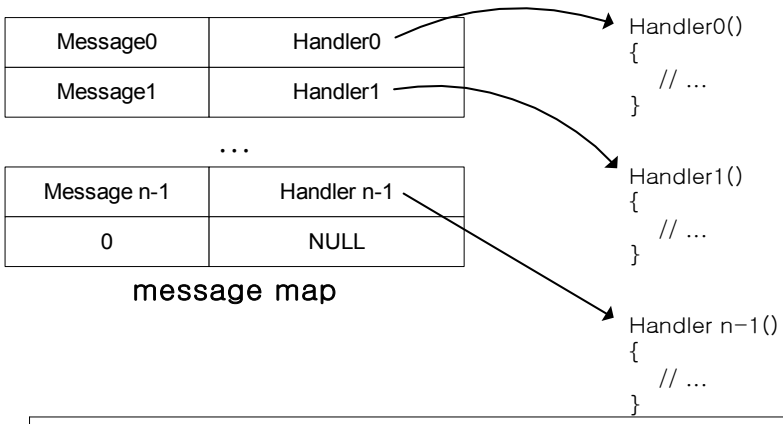
```
//Global object-----------------------------------------
CView app;

//Window procedure--------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam) {
    switch (iMsg) {
    case WM_CREATE:
        pCObject->OnCreate();
        return 0;
    case WM_PAINT:
        pCObject->OnDraw();
        return 0;
    case WM_DESTROY:
        pCObject->OnDestroy();
        return 0;
    case WM_LBUTTONDOWN:
        pCObject->OnLButtonDown();
        return 0;
    }//switch
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow) {
    pCObject = &app;
    pCObject->InitInstance(hInstance,szCmdLine,iCmdSh
    pCObject->Run();
    return pCObject->ExitInstance();
}//WinMain
```

이 구현의 문제점은 200여개가 넘는 메시지에 대응하는 가상 함수가 존재해야 한다는 것이다. 또한 추가되는 메시지와 사용자 정의 메시지는 처리할 수 없다. 우리는 이러한 구조가 아니라, 원하는 메시지만을 처리할 수 있는 좀 더 유연한 구조를 원한다. 단계 3에서 그 내용을 살펴보자.

단계 3: 멤버 함수 포인터 테이블의 이용

<절도비라>
이 절에서는 단계 2의 불완전한 해결책을 완전하게 해결하기 위해 멤버 함수에 대한 포인터 테이블을 이용하는 방법을 살펴본다. 이 테이블은 윈도우 메시지 ID와 메시지 ID에 대응하는 멤버 함수의 시작 주소를 가진다. 이 테이블을 메시지 맵이라 한다.
</절도비라>

멤버 함수를 가리키는 테이블을 이용하면 원하는 메시지만을 처리하는 윈도우 프로시저를 제작할 수 있다. 이 구조는 추가되는 메시지에 대한 자동화도 덩달아 가능하다. 이 부분이 MFC 메시지 맵의 핵심이다!



[그림 6.5] 메시지 맵(message map): 윈도우 메시지와 해당 메시지가 발생했을 때 호출할 핸들러의 시작 주소를 가지는 구조체 배열을 메시지 맵이라 한다.

메시지 맵은 구조체 배열(array of structure)이다. 구조체는 윈도우 메시지와 메시지에 대응하는 멤버 함수의 시작 주소를 가진다. 그리고 이 배열의 제일 끝에는 센티널(sentinel)로 메시지의 끝임을 알리는 특별한 상수 값을 넣어 둔다. 센티널은 가변적인 크기의 배열에서 탐색을 종료해야 할 시점을 가리킨다.

이제 MessagMap 구조체를 다음과 같이 만들 수 있다.

```
class CView;
typedef void (CView::*CViewFunPointer)();
typedef struct tagMessageMap {
    UINT iMsg;
    CViewFunPointer fp;
} MessageMap;
```

구조체의 첫 번째 멤버는 윈도우 메시지를 의미하며, 두 번째 멤버는 CView 클래스의 멤버 함수의 시작 주소를 가진다.

CView 클래스에서 메시지 맵은 다음과 같이 스태틱(static)으로 선언된다.

```
class CView : public CObject {
public:
    static MessageMap messageMap[];
public:
    void OnCreate();
    void OnDraw();
    void OnDestroy();
};//class CView
```

그리고 클래스의 구현 파일에서 다음과 같이 초기화한다.

```
//{{BEGIN_MESSAGE_MAP
MessageMap CView::messageMap[]={
    {WM_CREATE,CView::OnCreate},
    {WM_PAINT,CView::OnDraw},
    {WM_DESTROY,CView::OnDestroy},
    {0,NULL} // sentinel
};
//}}END_MESSAGE_MAP
```

구조체의 마지막에 추가된 센티널에 주목하라. 센티널은 탐색의 끝을 알리기 위해서 반드시 필요하다.

이제 윈도우 프로시저는 CView에 선언된 정적 메시지 맵 테이블을 검색하여 해당 메시지가 발견된 경우, 메시지에 대응하는 핸들러 - CView의 멤버 함수 -를 호출하도록 설계한다.

```
static CViewFunPointer fpCViewGlobal;
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam) {
    int i=0;

    while (CView::messageMap[i].iMsg!=0) {
        if (iMsg==CView::messageMap[i].iMsg) {
            fpCViewGlobal=CView::messageMap[i].fp;
```

```
            (app.*fpCViewGlobal)();
            return 0;
        }//if
        ++i;
    }//while
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc
```

처리할 메시지의 크기에 상관없이 윈도우 프로시저가 동작하는 것에 주목하라. 이제 메시지의 수에 상관없이 윈도우 프로시저는 변하지 않으며, 이렇게 설계된 코드에서 윈도우 프로시저를 미리 만들어 제공할 수 있고, 최종 사용자는 윈도우 프로시저의 감춰진 구현에 신경 쓰지 않아도 된다!

소스를 [예제 6.4]에 리스트하였다.

[예제 6.4] 단계 3 멤버 함수 포인터 테이블의 이용

```
#include <windows.h>

//Forward declaration-------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam);
//class CObject;
//typedef void (CObject::*CObjectFunPointer)();

//static CObjectFunPointer fpCObjectGlobal;
//CObject* pCObject;

//Class CObject-------------------------------------------
class CObject {
protected:
    static char szAppName[];
    HWND        hwnd;
    MSG         msg;
    WNDCLASSEX  wndclass;
public:
    void InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();
};//class CObject
```

```
void CObject::InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                           int iCmdShow) {
    wndclass.cbSize         =sizeof(wndclass);
    wndclass.style          =CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    =WndProc;
    wndclass.cbClsExtra     =0;
    wndclass.cbWndExtra     =0;
    wndclass.hInstance      =hInstance;
    wndclass.hIcon          =LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        =LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  =(HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   =NULL;
    wndclass.lpszClassName  =szAppName;
    wndclass.hIconSm        =LoadIcon(NULL,IDI_APPLICATION);

    RegisterClassEx(&wndclass);

    hwnd=CreateWindow(szAppName,     //window class name
        "The Hello Program",         //window caption
        WS_OVERLAPPEDWINDOW,         //window style
        CW_USEDEFAULT,               //initial x position
        CW_USEDEFAULT,               //initial y position
        CW_USEDEFAULT,               //initial x size
        CW_USEDEFAULT,               //initial y size
        NULL,                        //parent window handle
        NULL,                        //window menu handle
        hInstance,                   //program instance handle
        NULL);                       //creation parameters

    ShowWindow(hwnd,iCmdShow);
    UpdateWindow(hwnd);
}//CObject::InitInstance

void CObject::Run() {
    while (GetMessage(&msg,NULL,0,0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }//while
}//CObject::Run

WPARAM CObject::ExitInstance() {
    return msg.wParam;
}//CObject::ExitInstance
```

```
char CObject::szAppName[]="HelloWin";

//class CView-----------------------------------------
class CView;
typedef void (CView::*CViewFunPointer)();
typedef struct tagMessageMap {
    UINT iMsg;
    CViewFunPointer fp;
} MessageMap;
static CViewFunPointer fpCViewGlobal;
//CView* pCView;

class CView : public CObject {
public:
    static MessageMap messageMap[];
public:
    void OnCreate();
    void OnDraw();
    void OnDestroy();
};//class CView

//{{BEGIN_MESSAGE_MAP
MessageMap CView::messageMap[]={
    {WM_CREATE,CView::OnCreate},
    {WM_PAINT,CView::OnDraw},
    {WM_DESTROY,CView::OnDestroy},
    {0,NULL}
};
//}}END_MESSAGE_MAP

//CView Event handler---------------------------------------
void CView::OnCreate() {
}//CView::OnCreate

void CView::OnDraw() {
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    hdc=BeginPaint(hwnd,&ps);
    GetClientRect(hwnd,&rect);
    DrawText(hdc,"Hello, Windows!",-1,&rect,
```

```
        DT_SINGLELINE|DT_CENTER|DT_VCENTER);
    EndPaint(hwnd,&ps);
}//CView::OnDraw

void CView::OnDestroy() {
    PostQuitMessage(0);
}//CView::OnDestroy

//Global object----------------------------------------
CView app;

//Window procedure-------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam) {
    int i=0;

    while (CView::messageMap[i].iMsg!=0) {
        if (iMsg==CView::messageMap[i].iMsg) {
            fpCViewGlobal=CView::messageMap[i].fp;
            (app.*fpCViewGlobal)();
            return 0;
        }//if
        ++i;
    }//while
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow) {
    app.InitInstance(hInstance,szCmdLine,iCmdShow);
    app.Run();
    return app.ExitInstance();
}//WinMain
```

위의 구현은 MFC와 거의 비슷한 메시지 맵 구조를 가진다. WinMain()과 윈도우 프로시저는 이미 자동화 되었다. 이제 CView의 메시지 맵에만 집중하여 프로그램할 수 있다!

단계 4: 매크로의 사용

<절도비라>
이 절에서는 단계 3의 반복되는 코드를 간단하게 표현하고, 코드 생성을 쉽게 하기 위해서 매크로를 사용하는 방법을 이해하고, 실제 구현해보자.
</절도비라>

단계 3까지의 코드에서 이미 만들어진 WinMain()과 윈도우 프로시저 그리고, CObject를 사용할 수 있다는 것을 알았다. 최종 사용자는 단지 CObject를 상속받는 자신의 클래스를 정의한 다음 처리를 원하는 메시지에 대해서만 다음의 네 가지를 해 주어야 한다.

1) 정적 구조체 메시지 맵의 선언
2) 처리를 원하는 메시지 핸들러의 선언
3) 정적 구조체 메시지 맵의 초기화
4) 선언된 메시지 핸들러의 구현

MFC의 **어플리케이션 위저드(AppWizard)**는 사용자가 입력한 프로젝트 이름에 대해 위의 네 가지를 지원하는 코드를 생성한다. 그리고, **클래스 위저드(Class Wizard)**는 특별하게 처리된 주석을 이용하여 처음 세가지에 대한 코드를 추가/삭제한다. 네 번째 메시지 핸들러의 몸체 구현은 사용자에게 맡겨진다.

코드 생성을 쉽게 하기위해 다음과 같은 매크로를 정의할 수 있다.

```
#define DECLARE_MESSAGE_MAP()          static MessageMap messageMap[];
#define BEGIN_MESSAGE_MAP(class_name)\
        MessageMap class_name::messageMap[]={
#define END_MESSAGE_MAP()               {0,NULL}};
```

DECLARE_MESSAGE_MAP은 클래스에 정적 메시지 맵 배열을 선언하는 부분이다.

BEGIN_MESSAGE_MAP은 클래스 이름에 해당하는 메시지 맵 구조체를 초기화하기 위한 첫 부분이고, END_MESSAGE_MAP은 구조체 초기화의 마지막에 해당하는 부분이다. 이제 메시지 맵의 초기화 문장을 이 두 매크로 안에 넣는다. CView는 다음과 같이 구현한다.

```
class CView : public CObject {
public:
    // 1) you must properly declare message handlers
    //{{AFX_MESSAGEMAP
    void OnCreate();
    void OnDraw();
    void OnDestroy();
    //}}AFX_MESSAGEMAP

    // 2) you must include this macro in your class header
    DECLARE_MESSAGE_MAP()
};//class CView

// 3) next, you add your event handler message map, it's array of
//function pointer
BEGIN_MESSAGE_MAP(CView)
    {WM_CREATE,CView::OnCreate},
    {WM_PAINT,CView::OnDraw},
    {WM_DESTROY,CView::OnDestroy},
END_MESSAGE_MAP()
```

주석 처리된 //{{AFX와 //}}AFX에 주목하라. 이러한 특별한 주석은 후에 클래스 위저드에 의해 관리되는 코드를 구분하기 위해 사용된다. 소스를 [예제 6.5]에 리스트하였다.

[예제 6.5] 단계 4 매크로의 사용

```
#include <windows.h>

#define DECLARE_MESSAGE_MAP()          static MessageMap messageMap[];
#define BEGIN_MESSAGE_MAP(class_name)  MessageMap \
        class_name::messageMap[]={
#define END_MESSAGE_MAP()              {0,NULL}};

//Forward declaration----------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam);

//Class CObject----------------------------------------------
class CObject {
```

```
protected:
    static char szAppName[];
    HWND        hwnd;
    MSG         msg;
    WNDCLASSEX  wndclass;

public:
    void InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();
    // message handler isn't needed. Wow!
};//class CObject

void CObject::InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                           int iCmdShow) {
    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL,IDI_APPLICATION);

    RegisterClassEx(&wndclass);

    hwnd = CreateWindow(szAppName,    //window class name
        "The Hello Program",        //window caption
        WS_OVERLAPPEDWINDOW,        //window style
        CW_USEDEFAULT,              //initial x position
        CW_USEDEFAULT,              //initial y position
        CW_USEDEFAULT,              //initial x size
        CW_USEDEFAULT,              //initial y size
        NULL,                       //parent window handle
        NULL,                       //window menu handle
        hInstance,                  //program instance handle
        NULL);                      //creation parameters
```

```
    ShowWindow(hwnd,iCmdShow);
    UpdateWindow(hwnd);
}//CObject::InitInstance

void CObject::Run() {
    while (GetMessage(&msg,NULL,0,0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }//while
}//CObject::Run

WPARAM CObject::ExitInstance() {
    return msg.wParam;
}//CObject::ExitInstance

char CObject::szAppName[]="HelloWin";

// MessageMap, It's punch line! ---------------------------------
class CView;
typedef void (CView::*CViewFunPointer)();
typedef struct tagMessageMap {
    UINT iMsg;
    CViewFunPointer fp;
} MessageMap;
static CViewFunPointer fpCViewGlobal;//pointer to a member function

// class CView--------------------------------------------
class CView : public CObject {
public:
    // 1) you must properly declare message handlers
    //{{AFX_MESSAGEMAP
    void OnCreate();
    void OnDraw();
    void OnDestroy();
    //}}AFX_MESSAGEMAP

    // 2) you must include this macro in your class header
    DECLARE_MESSAGE_MAP()
};//class CView

// 3) next, you add your event handler message map, it's array of
// function pointer
BEGIN_MESSAGE_MAP(CView)
```

```
    {WM_CREATE,CView::OnCreate},
    {WM_PAINT,CView::OnDraw},
    {WM_DESTROY,CView::OnDestroy},
END_MESSAGE_MAP()

//CView Event handler----------------------------------------
void CView::OnCreate() {
}//CView::OnCreate

void CView::OnDraw() {
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    hdc = BeginPaint(hwnd,&ps);

    GetClientRect(hwnd,&rect);
    DrawText(hdc,"Hello, Windows!",-1,&rect,
        DT_SINGLELINE|DT_CENTER|DT_VCENTER);

    EndPaint(hwnd,&ps);
}//CView::OnDraw

void CView::OnDestroy() {
    PostQuitMessage(0);
}//CView::OnDestroy

//Global object---------------------------------------
CView app;

//Window procedure------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam) {
    int i=0;

    while ( 0 != CView::messageMap[i].iMsg ) {
        if ( iMsg == CView::messageMap[i].iMsg ) {
            fpCViewGlobal = CView::messageMap[i].fp;
            (app.*fpCViewGlobal)();
            return 0;
        }//if
        ++i;
    }//while
```

```
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow) {
    app.InitInstance(hInstance,szCmdLine,iCmdShow);
    app.Run();
    return app.ExitInstance();
}//WinMain
```

단계 5: 완성!

<절도비라>

이제 단계 4까지 완성한 소스를 기능별로 구분하여 별도의 소스로 작성하고, 반복되는 부분을 하나의 소스로 관리하는 방법을 살펴보자.

</절도비라>

이제 우리는 특별한 매크로와 정적 배열을 이용하여, WinMain()과 윈도우 프로시저는 숨겨진 채, 처리해야 할 메시지에만 집중하여 클래스를 설계할 수 있는 방법을 알았다. 이것이 MFC의 메시지 맵 설계 철학 중 하나이다. 공통된 부분은 MFC에 의해 제공되고, 우리는 처리해야 할 클래스에 집중해서 프로그래밍 한다!

이제 완벽하지는 않지만, MFC에 의해 미리 제공된다고 가정되는 부분과 그렇지 않은 부분으로 파일을 나누어 보자. 프로젝트는 다음과 같이 6개의 파일로 구성된다.

1) stdafx.h, stdafx.cpp
2) CObject.h, CObject.cpp
3) CView.h, CView.cpp

이 중 처음 네 파일은 MFC가 미리 제공하는 것이므로, 우리는 마지막 두 파일에만 집중해서 프로그래밍 한다. 마지막 두 파일에는 WinMain()과 윈도우 프로시저에 대한 어떠한 정보도 없으며 단지 메시지와 메시지 핸들러 만을 담고 있다. 소스를 아래에 리스트하였다.

[예제 6.6] stdafx.h

```
#include <windows.h>

#ifndef _stdafx_
#define _stdafx_

#define DECLARE_MESSAGE_MAP()           static MessageMap messageMap[];
#define BEGIN_MESSAGE_MAP(class_name)   MessageMap \
        class_name::messageMap[]={
#define END_MESSAGE_MAP()               {0,NULL}};

//Forward declaration-------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam);

#endif _stdafx_
```

[예제 6.7] stdafx.cpp

```
#include <windows.h>
#include "CView.h"

extern CView app;

LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam) {
    int i = 0;

    while ( CView::messageMap[i].iMsg != 0 ) {
        if ( iMsg == CView::messageMap[i].iMsg ) {
            fpCViewGlobal=CView::messageMap[i].fp;
            (app.*fpCViewGlobal)(wParam,lParam);
            return 0L;
        }//if
        ++i;
    }//while
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine,int iCmdShow) {
```

```
    app.InitInstance(hInstance,szCmdLine,iCmdShow);
    app.Run();
    return app.ExitInstance();
}//WinMain
```

[예제 6.8] CObject.h

```
#include <windows.h>

#ifndef _CObject_
#define _CObject_

class CObject {
protected:
    static char szAppName[];
    HWND        hwnd;
    MSG         msg;
    WNDCLASSEX  wndclass;

public:
    void InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();
};//class CObject

#endif
```

[예제 6.9] CObject.cpp

```
#include <windows.h>
#include "stdafx.h"
#include "CObject.h"

void CObject::InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                           int iCmdShow) {
    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
```

```
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL,IDI_APPLICATION);

    RegisterClassEx(&wndclass);

    hwnd = CreateWindow(szAppName,  //window class name
        "The Hello Program",        //window caption
        WS_OVERLAPPEDWINDOW,        //window style
        CW_USEDEFAULT,              //initial x position
        CW_USEDEFAULT,              //initial y position
        CW_USEDEFAULT,              //initial x size
        CW_USEDEFAULT,              //initial y size
        NULL,                       //parent window handle
        NULL,                       //window menu handle
        hInstance,                  //program instance handle
        NULL);                      //creation parameters

    ShowWindow(hwnd,iCmdShow);
    UpdateWindow(hwnd);
}//CObject::InitInstance

void CObject::Run() {
    while (GetMessage(&msg,NULL,0,0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }//while
}//CObject::Run

WPARAM CObject::ExitInstance() {
    return msg.wParam;
}//CObject::ExitInstance

char CObject::szAppName[] = "HelloWin";
```

[예제 6.10] CView.h

```
#include <windows.h>
#include "stdafx.h"
#include "CObject.h"

#ifndef _CView_
#define _CView_

// message map structure ------------------------------------------
class CView;

typedef LRESULT (CView::*CViewFunPointer)(WPARAM,LPARAM);

typedef struct tagMessageMap {
    UINT iMsg;
    CViewFunPointer fp;
} MessageMap;

static CViewFunPointer fpCViewGlobal;//pointer to a member function

// class CView ----------------------------------------------------
class CView : public CObject {
public:
    //{{AFX_MESSAGE
    LRESULT OnCreate(WPARAM,LPARAM);
    LRESULT OnDraw(WPARAM,LPARAM);
    LRESULT OnDestroy(WPARAM,LPARAM);
    LRESULT OnLButtonDown(WPARAM,LPARAM);
    //}}AFX_MESSAGE

    DECLARE_MESSAGE_MAP()
};//class CView

#endif
```

[예제 6.11] CView.cpp

```
#include <windows.h>
#include "stdafx.h"
#include "CView.h"
```

```
CView app; // one global object

//{{AFX_MESSAGE
BEGIN_MESSAGE_MAP(CView)
    {WM_CREATE,CView::OnCreate},
    {WM_PAINT,CView::OnDraw},
    {WM_DESTROY,CView::OnDestroy},
    {WM_LBUTTONDOWN,CView::OnLButtonDown},
END_MESSAGE_MAP()
//}}AFX_MESSAGE

LRESULT CView::OnCreate(WPARAM wParam,LPARAM lParam) {
    return 0L;
}//CView::OnCreate

LRESULT CView::OnDraw(WPARAM wParam,LPARAM lParam) {
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    hdc = BeginPaint(hwnd,&ps);

    GetClientRect(hwnd,&rect);
    DrawText(hdc,"Hello, Windows!",-1,&rect,
        DT_SINGLELINE|DT_CENTER|DT_VCENTER);

    EndPaint(hwnd,&ps);
    return 0L;
}//CView::OnDraw

LRESULT CView::OnDestroy(WPARAM wParam,LPARAM lParam) {
    PostQuitMessage(0);
    return 0L;
}//CView::OnDestroy

LRESULT CView::OnLButtonDown(WPARAM wParam,LPARAM lParam) {
    MessageBox(hwnd, "LButtonDown", "WM_LBUTTONDOWN", MB_OK);
    return 0L;
}//CView::OnCreate
```

MFC가 생성한 코드에 CView.h와 CView.cpp만 있다고 생각해 보라. 처음에는 WinMain()과 윈도우 프로시저가 없는 코드에 당황하겠지만, 원리를 이해한 후 이것은 간결함 그 자체이다!

대리자(델리게이트, delegate)

메시지 맵을 각 클래스의 정적 변수로 포함하지 않고, 구현할 수 없을까? 그것을 **동적 메시지 맵(dynamic message map)**이라 부르자. 동적 메시지 맵의 구현은 다음과 같은 이유 때문에 구현불가능하게 보인다.

- 멤버 함수에 대한 포인터는 클래스에 종속적이다. 예를 들면, 클래스 A의 멤버 함수 int A::Fun(int, float)에 대한 포인터는 클래스 B의 멤버 함수 int B::Fun(int, float)와 호환되지 않는다.

위의 사실을 이상하게 생각할 수 있다. 멤버 함수는 함수의 첫 번째 파라미터로 객체의 시작 주소인 this를 받는 것 말고, 일반 함수와 같은 것이 아닌가? 그렇기는 하다. 하지만 this때문에 문제가 된다. 예를 들어 아래와 같은 소스를 고려해 보자.

```
#include <stdio.h>

class KBase
{
public:
    int m_aiData[10];

    void BaseFun(int i, float f)
    {
        printf( "KBase::BaseFun() this=%p\n", this );
    }
};

class KMiddle : public KBase
{
public:
    int m_aiData[10];
};
```

```
class KMiddle2 : public KBase
{
public:
    int m_aiData[20];
};

class KDerived : public KMiddle, public KMiddle2
{
public:
    int m_aiData[30];
};

void main()
{
    KDerived    d;

    d.BaseFun( 1, 2.f );
}//main()
```

위 소스는 컴파일되지 않는다. 왜냐하면 main()에 사용된 d.BaseFun()이 모호하기 때문이다. 즉, BaseFun()이 호출되었을 때 this가 KMiddle의 시작 주소인지, KMiddle2의 시작 주소인지 알 수 있는 방법이 없기 때문이다. 이제 main()을 다음과 같이 고쳐서 컴파일해 보자.

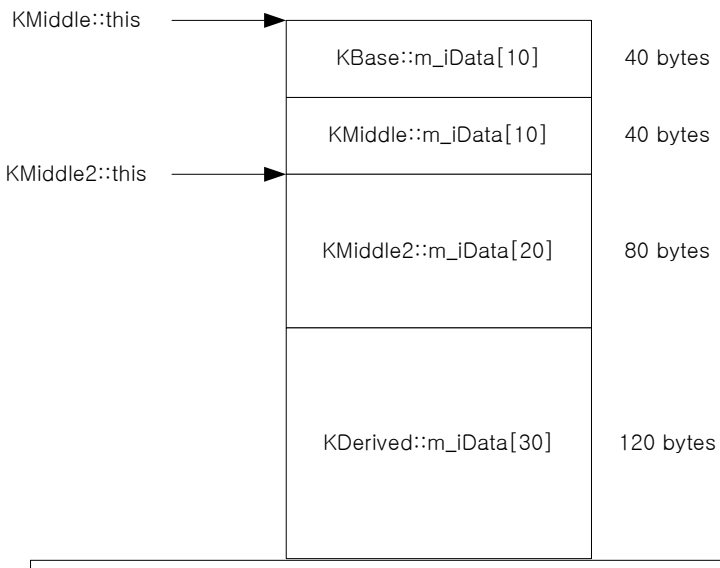
```
...
void main()
{
    KDerived    d;

//    d.BaseFun( 1, 2.f );
    d.KMiddle::BaseFun( 1, 2.f );
    d.KMiddle2::BaseFun( 1, 2.f );
}//main()
```

실행 결과는

```
KBase::BaseFun() this=0012FE44
KBase::BaseFun() this=0012FE94
```

이다. 두 번째 값이 첫 번째 값보다 80만큼 큰 것을 알 수 있는데 그것은 객

체 d가 다음 그림과 같이 메모리 할당되었기 때문이다.



[그림 6.6] 다중 상속의 경우 같은 문맥에서 this는 여러 값으로 해석될 수 있다.

KMiddle2의 멤버 함수가 실행 중일 때는 KMiddle의 값을 볼 수 없다는 것을 주의하자. 그러므로 컴파일러는 KMiddle2를 통해 접근한 멤버 함수에 대해, 실제 객체의 시작 주소(original this)에서 KMiddle2의 데이터가 시작하는 상대 위치 80을 더한 값을 this로 취하는 것이다. 이것은 멤버 함수에 대한 포인터 변수가 단순히 멤버 함수의 시작 주소 4바이트를 가지는 것 외에 this에 더해 주어야 하는 상대 위치 값 4바이트(여기서는 80)를 더 가짐을 의미한다. 그렇다면 멤버 함수에 대한 포인터 변수는 8바이트일까? 그렇지 않다!

우리는 또한 가상 함수를 고려해 주어야 한다. 가상함수에 대한 고려는 가상 함수 테이블의 시작 주소 4바이트와, 상대 위치 값 4바이트가 추가됨을 의미한다. 그러므로 멤버 함수에 대한 포인터는 최악의 경우에 16바이트가 될 수 있다.

C++ 표준은 멤버 함수의 동작에 대한 명세만 있을 뿐, 구현에 대한 명세가 없으므로 멤버 함수의 데이터 크기는 컴파일러 구현자에 따라 각각 모두 다르다. 비주얼 C++의 경우, 단일 상속인 경우는 4바이트, 다중 상속인 경우는 8바이트, 가상함수가 포함된 경우는 16바이트가 된다. 그러므로 한 멤버 함수에 대한 포인터를 void*에 대입할 수 없을 뿐 아니라, 다른 멤버 함수에 대한 포인터로 형 변환할 수 없다(강제로 한다면, 중요한 데이터를 잃어버리게 된다).

필자는 **델리게이트(delegate)**, 즉 **일반 멤버 함수 포인터(generic**

member function pointer) 가 C++ 표준에 추가될 것을 기대한다. 비주얼 C++ 7.x의 경우 델리게이트를 지원한다. 하지만, 이것은 표준 C++이 아니다.

한 가지 해결책은 델리게이트를 지원하는 기능을 직접 구현하는 것이다. 이것은 템플릿 부분 특화(partial template specialization)에 의해 빠르게 구현될 수 있다. **코드 프로젝트(www.codeproject.com)**에서 delegate로 검색해서 문서를 찾아보기 바란다. 또 다른 구현은 **부스터 라이브러리(boost library, www.boost.org)**에서 찾아 볼 수 있다. 상세한 설명은 코드 프로젝트의 문서와 부스터 라이브러리의 문서를 꼭 읽어 보고, 필자가 느꼈던 코드에 대한 놀라움을 같이 느낄 수 있게 되기를 바란다.

요약

우리는 이 장에서 Win32를 이용해서 메시지 맵으로 동작하는 Hello 프로그램을 단계별로 만들었다. 메시지 맵은 MFC 프로그래밍의 핵심일 뿐만 아니라, 모든 윈도우 프로그래밍의 핵심이라고 할 수 있다.

- MFC는 메시지 루프를 **메시지 펌프(message pump)**라고 부른다. MFC의 메시지 펌프는 GetMessage()를 사용하지 않고 PeekMessage()를 사용하며, 노는 시간(idle time)을 처리하기 위한 특별한 가상 함수 OnIdle()을 가진다.
- **메시지 맵**이란 윈도우 메시지를 받아야 할 클래스가 가지는 정적 구조체 배열이다. 이 정적 구조체 배열은 처리할 메시지와 메시지에 대응하는 멤버 함수에 대한 포인터를 핵심적으로 가진다.
- MFC는 메시지 맵의 코드 자동화를 위해 **DECLARE_MESSAGE_MAP** 등의 매크로를 사용한다.

[문서의 끝]