□ 예제 프로그램 1: LabProject00(윈도우 프로그램 생성하기)

DirectX SDK를 사용하여 게임 프로그램을 작성하기 전에 게임 프로그램의 기본 골격을 만들어 보자. 복잡한 게임 프로그램을 작성하기 전에 DirectX SDK를 사용한 간단한 기본 윈도우 프로그램을 작성해보도록 하자. 먼저 Visual Studio의 응용프로그램 마법사를 사용하여 간단한 윈도우 프로그램을 만들어 보자. 생성된 새로운 윈도우 응용프로그램을 기반으로 앞으로 사용하게 될 게임 프로그램의 골격을 만들어 보자.

① 응용 프로그램 마법사를 통한 새로운 응용 프로그램 생성

❶ 프로젝트 "새로 만들기" 메뉴

다음 <그림 1>과 같이 Visual Studio 메뉴 바의 『파일』 메뉴에서 『새로 만들기』, 『프로젝트』 메뉴 항목을 차례로 선택한다. 그러면 <그림 2>와 같은 "새 프로젝트" 대화상자가 나타난다.

☞『파일』 메뉴 ▶『새로 만들기』▶『프로젝트』

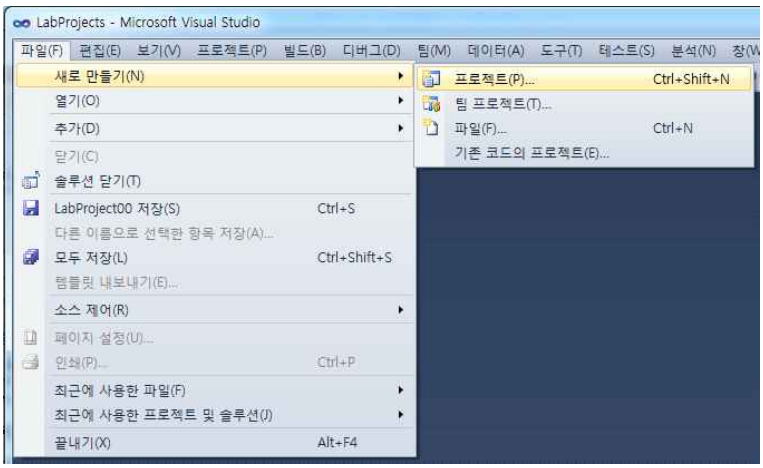

그림 1. 프로젝트 만들기

❷ "새 프로젝트" 대화상자

"새 프로젝트" 대화상자에서 "설치된 템플릿"에서 "Win32"를 선택하고 "Win32 프로젝트"를 클릭한다. 그리고 프로젝트 "이름"을 "**LabProject00**"으로 입력한다. 『솔루션 이름』을 "LabProjects"로 입력한다. 프로젝트 "위치"를 설정한 후에 『확인』 버튼을 클릭한다(프로젝트 위치는 『찾아보기...』 버튼을 눌러서 폴더를 선택하여 설정할 수 있다). 그러면, "응용프로그램 마법사"가 <그림 3>과 같이 실행된다.

☞『설치된 템플릿』▶『Win32』▶『Win32 프로젝트』▶
『이름』▶ "LabProject00" ▶『솔루션 이름』▶ "LabProjects" ▶『찾아보기...』▶『확인』

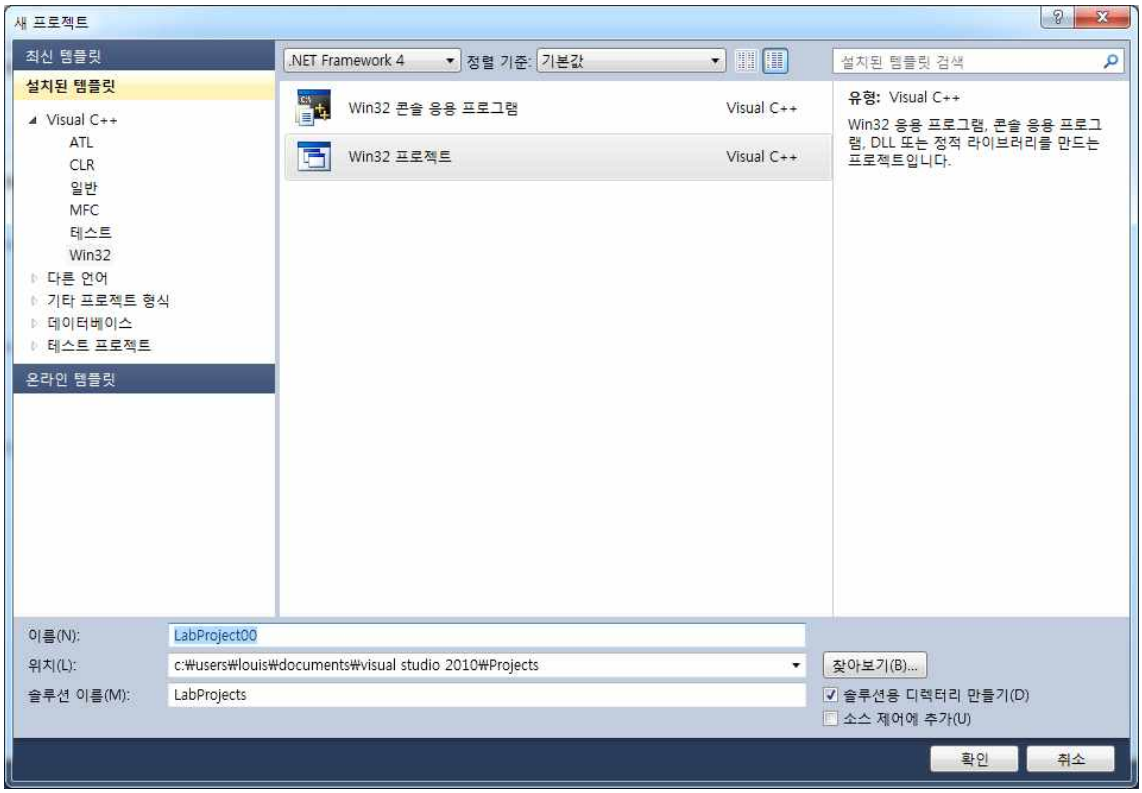

그림 2. 새 프로젝트 대화상자

❸ “응용 프로그램 마법사“

“응용 프로그램 마법사“에서 『마침』 버튼을 클릭한다. 그러면 새로운 윈도우 응용 프로그램을 위한 프로젝트가 생성된다. 생성된 프로젝트에는 Visual C++ 의 ”응용 프로그램 마법사”가 자동으로 생성한 몇 개의 소스 파일이 포함되어 있다. 이 소스 파일들은 <그림 4>의 『솔루션 탐색기』에서 보는 것과 같이 “리소스 파일”, “소스 파일”, “헤더 파일”로 구분되며 다음과 같다.

- LabProject00.h
- LabProject00.cpp
- stdafx.h
- stdafx.cpp
- Resource.h
- targetver.h

하지만 이 과정에서는 “LabProject00.cpp" 파일만을 자세히 살펴볼 것이다. 이후에 게임 프로그램의 골격을 구성하기 위해 계속적으로 새로운 내용을 이 프로젝트에 추가할 것이다.

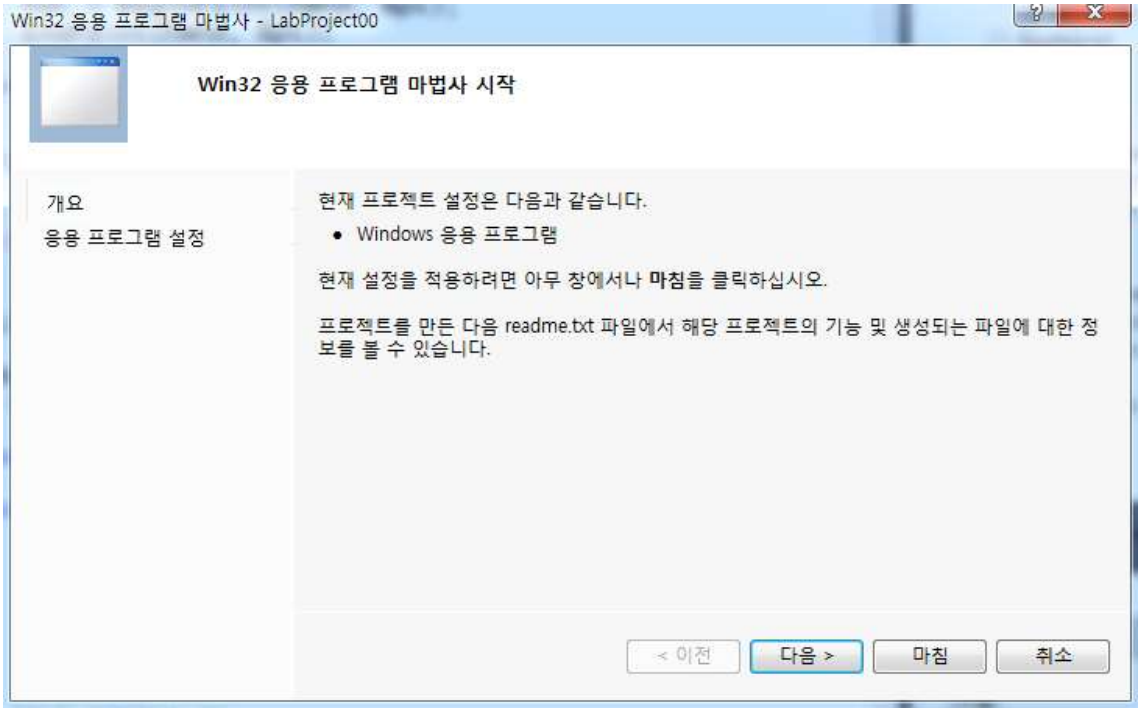

그림 3. 응용 프로그램 마법사

응용 프로그램 마법사가 생성한 프로젝트 리소스 파일의 내용은 대부분 그대로 사용할 것이며 거의 수정하지 않을 것이므로 여기에서는 자세히 다루지 않는다. 프로젝트의 소스 파일에서 “resource.h"는 윈도우의 리소스(메뉴, 대화상자 등)의 ID를 정의하는 헤더 파일이다. 여기에서는 윈도우 리소스를 다루지 않을 것 이므로 이 파일은 무시해도 된다. “targetver.h"는 응용프로그램의 플랫폼을 정의하는 헤더 파일이며 이 파일 역시 수정하지 않고 응용 프로그램 마법사가 생성한 그대로를 사용할 것이다. **”stdafx.h"는 프로그램에서 사용하는 주요 헤더 파일들을 포함한다. 나중에 이 파일에 Direct3D 12의 헤더 파일을 추가할 것이다.** “stdafx.cpp" 소스 파일은 사전컴파일헤더(Precompiled Header)를 생성하기 위한 것으로 수정하지 않고 그대로 사용할 것이다. “LabProject00.h" 헤더 파일은 리소스와 관련된 사용자 헤더 파일을 포함한다. 이 파일은 수정하지 않고 그대로 사용할 것이다.

“LabProject00.cpp" 소스 파일은 프로그램의 소스 코드를 포함하는 파일이다. 이 파일을 수정하여 게임 프로그램의 골격을 완성할 것이다. 이 파일에 게임 프로그램의 골격을 만들기 위해 추가되는 파일을 위한 헤더 파일들을 포함시킬 것이다.

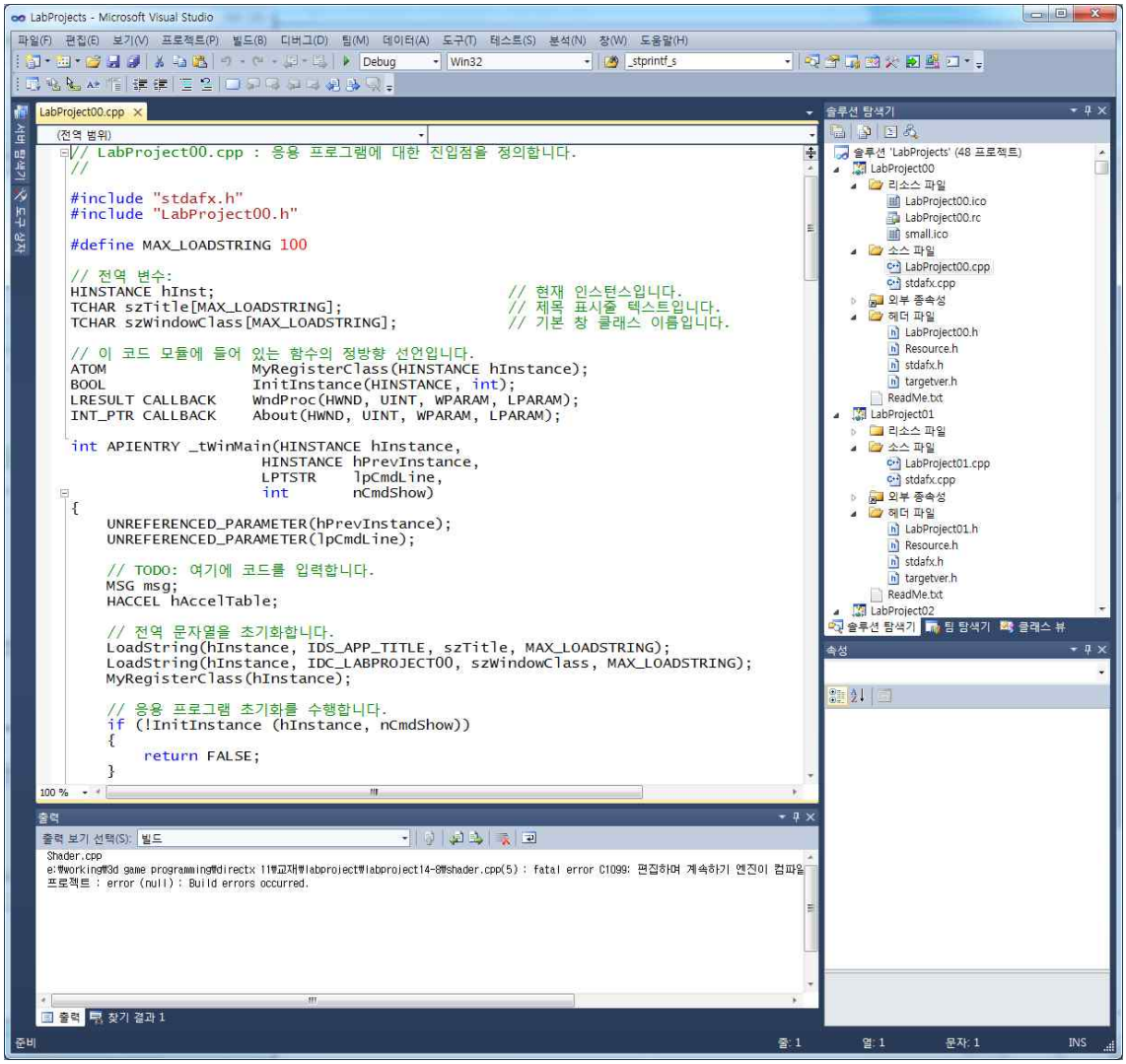

그림 4. 새로운 프로젝트

② 응용 프로그램 실행

❶ 생성된 응용 프로그램 빌드 및 실행

파일의 내용을 자세히 살펴보기 전에 생성된 프로젝트를 빌드하고 실행하여 보자. <그림 5>와 같이 『빌드』 메뉴 또는 <그림 6>과 같이 『디버그』 메뉴에서 프로젝트를 빌드(컴파일과 링크)할 수 있다. 보통 『디버그』 메뉴에서 『디버깅 시작』을 선택하거나 “F5” 키를 눌러서 프로젝트 빌드와 실행을 한꺼번에 처리한다.

☞ 『파일』 메뉴 ▶ 『빌드』 ▶ 『LabProject00 빌드』

☞ 『파일』 메뉴 ▶ 『디버그』 ▶ 『디버깅 시작』

그림 5. 빌드 메뉴

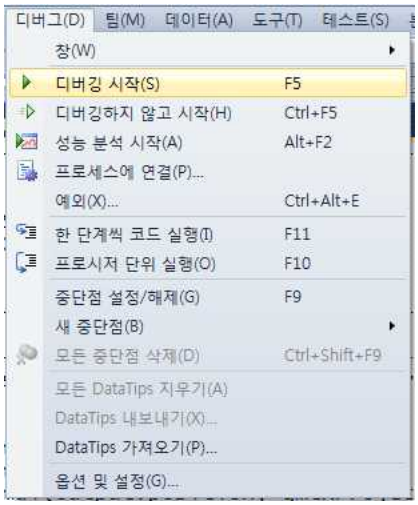

그림 6. 디버그 메뉴

프로젝트 실행 결과는 <그림 7>과 같다. 생성된 프로젝트는 윈도우 프로그램이 갖추어야 할 기본적인 내용의 소스 코드를 갖추고 있다. 앞으로 생성된 소스 코드를 분석해 보고 이것을 바탕으로 3D 그래픽 프로그램을 작성할 것이다.

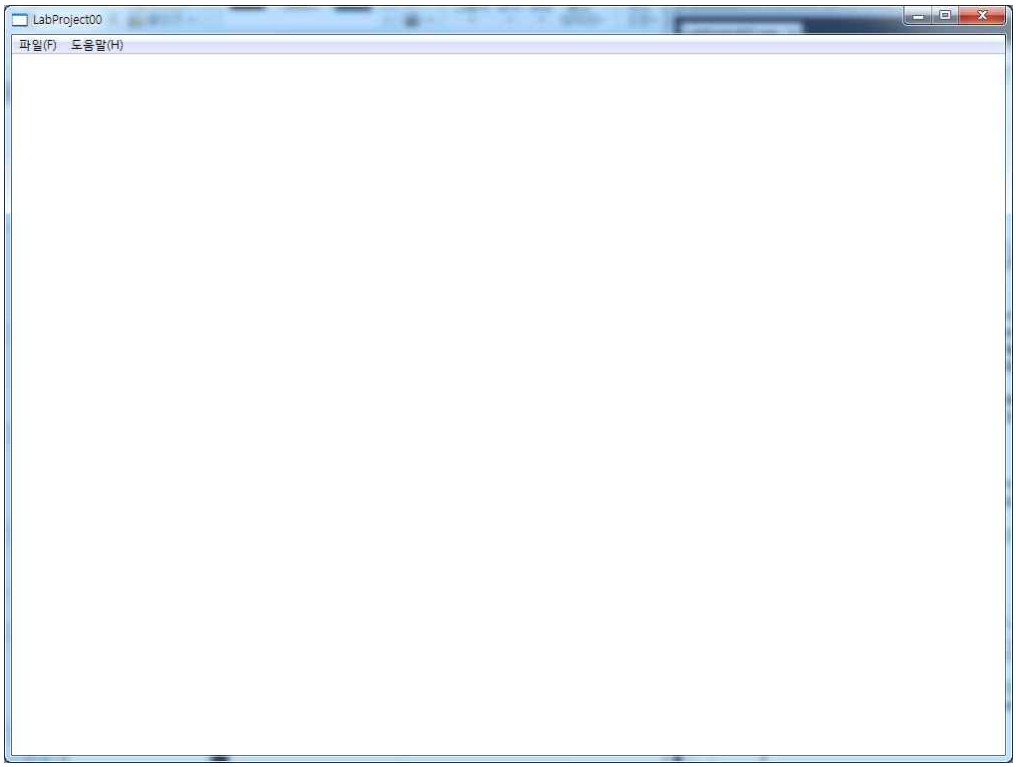

그림 7. 간단한 윈도우 프로그램

□ 윈도우 프로그램 분석하기

Visual Studio의 "응용프로그램 마법사"가 자동으로 생성한 "LabProject00.cpp" 파일을 분석하여 윈도우 프로그램의 기본 골격과 윈도우 프로그래밍 모델을 이해하도록 하자.

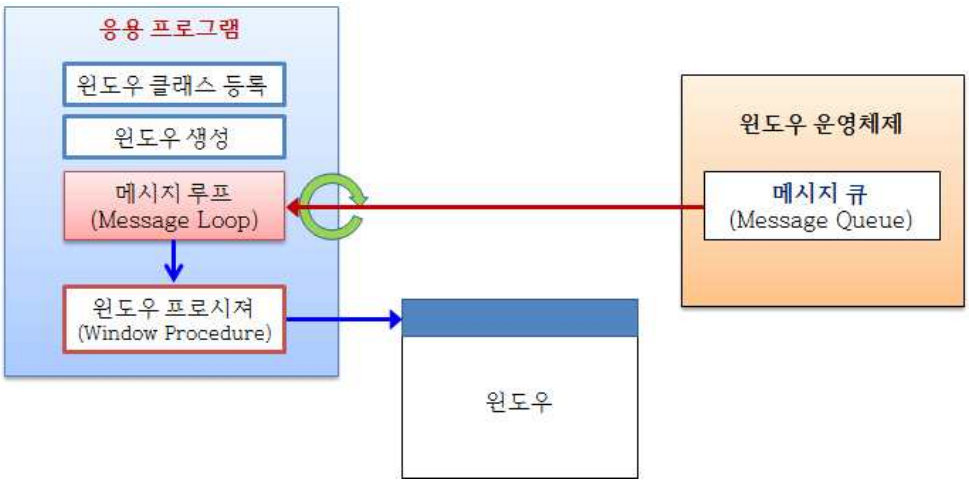

그림 8. 윈도우 메시지 처리

① WinMain 함수

윈도우 응용 프로그램의 시작 진입점(Main Entry Point)을 나타내는 함수이다. 윈도우 응용 프로그램이 실행되면 윈도우 운영체제는 이 함수를 실행하여 응용프로그램을 시작한다. 이 함수가 종료(반환)되면 응용 프로그램이 종료된다.

```
int APIENTRY _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR
lpCmdLine, int nCmdShow)
{
   UNREFERENCED_PARAMETER(hPrevInstance);
   UNREFERENCED_PARAMETER(lpCmdLine);

   MSG msg;
   HACCEL hAccelTable;

   // 전역 문자열을 초기화합니다.
   LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
   LoadString(hInstance, IDC_FIRST, szWindowClass, MAX_LOADSTRING);

   //❶ 윈도우 클래스를 등록한다.
   MyRegisterClass(hInstance);

   //❷ 응용프로그램 초기화(주 윈도우 생성)를 수행한다.
   if (!InitInstance(hInstance, nCmdShow))
   {
      return FALSE;
```

```
    }

    hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_FIRST));

    //❸ 기본 메시지 루프이다.
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}
```

일반적인 윈도우 응용 프로그램이 되려면 이 함수는 다음과 같은 기본적인 일을 처리해야 한다.

❶ 윈도우 클래스를 시스템에 등록

윈도우 클래스는 윈도우의 특성/모양(Class Style)과 동작 방식(Window Procedure)을 나타낸다. 윈도우 클래스를 만들어 윈도우 시스템에 등록을 하면 그 클래스의 윈도우를 생성할 수 있다. 그러므로 응용 프로그램에서 생성할 윈도우가 속하는 윈도우 클래스가 윈도우 시스템에 존재하지 않으면 반드시 새로운 윈도우 클래스를 윈도우 시스템에 등록해야 한다. 윈도우 클래스의 윈도우 프로시져는 윈도우 클래스에 속하는 모든 윈도우가 윈도우 메시지를 어떻게 처리하는 가를 나타내는 함수이다. 일반적으로 윈도우 응용 프로그램이 실행되면 응용 프로그램을 대표하는 주 윈도우(Main Window)가 나타나고 이 윈도우의 모양과 동작 방식은 다른 응용 프로그램과 차별화되어야 한다. 적어도 윈도우 프로시져는 달라야 할 것이다. 그러므로 윈도우 응용 프로그램은 최소한 주 윈도우가 속하는 윈도우 클래스를 시스템에 등록해야 한다. 윈도우 클래스를 등록하기 위한 윈도우 API 함수는 RegisterClassEx()이다.

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style= CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc= WndProc; //윈도우 프로시져를 설정한다.
    wcex.cbClsExtra= 0;
    wcex.cbWndExtra= 0;
    wcex.hInstance= hInstance;
    wcex.hIcon= LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LABPROJECT00));
    wcex.hCursor= LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground= (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName= MAKEINTRESOURCE(IDC_LABPROJECT00);
    //wcex.lpszMenuName= NULL; //메뉴를 없애려면
    wcex.lpszClassName= szWindowClass;
    wcex.hIconSm= LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
```

```
    return RegisterClassEx(&wcex);
}
```

❷ 응용 프로그램 초기화 및 주 윈도우 생성

다음은 주 윈도우를 생성하고 화면에 보이도록 하는 함수이다. 윈도우를 생성하는 윈도우 API 함수는 CreateWindow()이다.

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;

   hInst = hInstance; // 인스턴스 핸들을 전역변수에 저장합니다.

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
         CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}
```

❸ 메시지 루프

다음의 코드는 응용 프로그램의 메시지 큐에서 메시지를 가져와서(GetMessage) 메시지를 해당 윈도우에게 전달하는(DispatchMessage) 메시지 루프를 나타낸다. GetMessage() API 함수가 FALSE를 반환하는 경우는 메시지 큐에서 가져온 메시지가 WM_QUIT 메시지일 때이다. WM_QUIT 메시지는 응용 프로그램을 종료하는 경우에 발생한다. 다른 메시지의 경우에는 TRUE를 반환한다. 그러므로 다음의 메시지 루프는 WM_QUIT 메시지를 처리할 때까지 계속 반복하여 실행될 것이다.

```
   while (GetMessage(&msg, NULL, 0, 0))
   {
      if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
      {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }
```

② 윈도우 프로시져

다음은 윈도우 클래스를 등록할 때 설정한 윈도우 프로시져 함수이다. 이 함수는 주 윈도우가 처리해야 하는 윈도우 메시지가 주 윈도우에게 전달될 때 호출되는 메시지 처리 함수이다. 즉, 메시지 루프에서 DispatchMessage() 윈도우 API 함수를 호출하면 WndProc() 함수가 실행된다. 윈도우 프로시져는 메시지를 나타내는 4개의 파라메터(윈도우 핸들, 메시지 ID, 2개의 파라메터)를 매개변수를 가진다.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   int wmId, wmEvent;
   PAINTSTRUCT ps;
```

```
    HDC hdc;

    switch (message)
    {
        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // 메뉴의 선택영역을 구문분석합니다.
            switch (wmId)
            {
                case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);
            // TODO: 여기에 그리기 코드를 추가합니다.
            EndPaint(hWnd, &ps);
            break;
        case WM_DESTROY:
            PostQuitMessage(0); //WM_QUIT 메시지를 생성하여 메시지 큐에 삽입한다.
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

다음은 프로젝트 “LabProject00”의 LabProject00.cpp 파일의 전체 내용이다.

```
// LabProject00.cpp : 응용 프로그램에 대한 진입점을 정의합니다.

#include "stdafx.h"
#include "LabProject00.h"

#define MAX_LOADSTRING 100

// 전역변수:
HINSTANCE hInst;                                // 현재 인스턴스입니다.
TCHAR szTitle[MAX_LOADSTRING];                  // 제목 표시줄 텍스트입니다.
TCHAR szWindowClass[MAX_LOADSTRING];            // 기본창 클래스 이름입니다.

// 이 코드 모듈에 들어있는 함수의 정방향 선언입니다.
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR
lpCmdLine, int nCmdShow)
```

```
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: 여기에 코드를 입력합니다.
    MSG msg;
    HACCEL hAccelTable;

    // 전역 문자열을 초기화합니다.
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_FIRST, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // 응용프로그램 초기화를 수행합니다.
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_FIRST));

    // 기본 메시지루프입니다.
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}


///
//  함수: MyRegisterClass()
//  목적: 창클래스를 등록합니다.
//  설명:
//    Windows 95에서 추가된'RegisterClassEx' 함수보다 먼저 해당코드가 Win32
//    시스템과 호환되도록 하려는 경우에만 이 함수를 사용합니다. 이 함수를 호출해야
//    해당 응용프로그램에 연결된 '올바른 형식의' 작은 아이콘을 가져올 수 있습니다.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LABPROJECT00));
    wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
```

```
   //wcex.lpszMenuName = MAKEINTRESOURCE(IDC_LABPROJECT00);
   wcex.lpszMenuName = NULL;
   wcex.lpszClassName = szWindowClass;
   wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

   return RegisterClassEx(&wcex);
}

//
//   함수: InitInstance(HINSTANCE, int)
//   목적: 인스턴스 핸들을 저장하고 주창을 만듭니다.
//   설명:
//        이 함수를 통해 인스턴스 핸들을 전역변수에 저장하고
//        주프로그램창을 만든 다음 표시합니다.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;

   hInst = hInstance; // 인스턴스 핸들을 전역변수에 저장합니다.

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);
   if (!hWnd) return FALSE;

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}


//
//  함수: WndProc(HWND, UINT, WPARAM, LPARAM)
//  목적: 주창의 메시지를 처리합니다.
//  WM_COMMAND - 응용프로그램 메뉴를 처리합니다.
//  WM_PAINT   - 주창을 그립니다.
//  WM_DESTROY - 종료 메시지를 게시하고 반환합니다.
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   int wmId, wmEvent;
   PAINTSTRUCT ps;
   HDC hdc;

   switch (message)
   {
      case WM_COMMAND:
         wmId    = LOWORD(wParam);
         wmEvent = HIWORD(wParam);
         // 메뉴의 선택영역을 구문분석합니다.
         switch (wmId)
         {
            case IDM_ABOUT:
               DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
               break;
            case IDM_EXIT:
```

```
                DestroyWindow(hWnd);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        break;
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        // TODO: 여기에 그리기 코드를 추가합니다.
        EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// 정보 대화상자의 메시지 처리기입니다.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;
    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```