2013년도 1학기

10장 멀티 스레드

- 학습목표
  - 스레드 개념을 알고 멀티 스레드의 필요성을 이해할 수 있다.
  - 스레드를 생성하고 종료할 수 있다.
  - 멀티 스레드에서 스레드 간 동기화가 필요한 이유를 이해할 수 있다.
  - 멀티 스레드의 동기화를 위해 이벤트를 사용할 수 있다.

- 내용
  - 스레드 생성
  - 스레드 동기화

1절. 스레드 생성

- 프로세스 (Process): 실행중인 프로그램의 한 인스턴스
  - 운영 체제는 실행된 프로그램을 프로세스 단위로 관리한다.
  - 실제로 작업하는 담당은 스레드 (Thread)

- 스레드: 연속된 명령어를 차례로 처리하는데, 이렇게 연속적으로 처리되는 명령어의 흐름

- 단일 스레드의 문제점(예)
  1. 서버가 여러 클라이언트들의 요구를 처리해야 하는 경우 하나의 클라이언트의 요구를 처리하고 있는 동안 나머지 클라이언트들은 무한정 기다려야 함
  2. 화면상에 여러 객체의 움직임을 나타낼 때 한 객체에 대한 처리를 마친 후에야 다른 객체에 대한 처리 할 수 있음

- 문제점 해결
  - 멀티스레드: CPU가 각 스레드를 번갈아 가면서 처리하여 마치 각 스레드를 동시에 처리하는 것과 같은 효과를 나타냄

스레드 프로그래밍 단계

1. #include <process.h>
   - 스레드 생성 및 관리에 관한 함수가 정의

2. 비주얼 스튜디오 환경에서 멀티 스레드 DLL 셋팅
   - 프로젝트 환경에서 코드 생성시 멀티 스레드를 이용하도록 셋팅

스레드 생성 함수

스레드 생성 함수

```
uintptr_t _beginthreadex( // 스레드 핸들값 반환
    void *security,        // 구조체 SECURITY_ATTRIBUTES의 포인터 변수, NULL
    unsigned stack_size,   // 스레드를 위한 스택 크기, 0을 이용
    unsigned ( *start_address )( void * ), //스레드로 실행될 함수의 이름
    void *arglist,         // 스레드 함수에게 전달될 파라미터 공간의 주소
    unsigned initflag,     // 스레드 상태지정 값으로 실행시키기 위해서 0을 이용
    unsigned *thrdaddr     // 스레드 ID를 받기 위한 32비트 포인터 변수, NULL
);
```

사용 예)

```
HANDLE hThread;
hThread = (HANDLE)_beginthreadex (NULL, 0,
                (unsigned int(__stdcall *)(void *))ThreadProc, NULL, 0, NULL);
```

스레드 종료 및 정리 함수

- 스레드 종료

```
void _endthreadex(
    unsigned retval // 반환하기 원하는 값
);
```

- 스레드 정리

```
BOOL WINAPI CloseHandle(
    HANDLE hObject
);
```

10-1 멀티 스레드 기초

예제) 왼쪽 마우스 버튼을 눌러 스레드를 생성하여 사각형을 그린다.

```
#include <windows.h>
#include <process.h>
#include <time.h>

HWND   hwnd; // WinMain()함수의 지역변수를 전역변수로 전환

LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM
    wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpszCmdLine, int nCmdShow)
{
  . . . 생략 . . .
}
```

10-1 멀티 스레드 기초

```
void ThreadProc()   // 스레드 함수
{
   HDC hdc;
   int i;
   srand((unsigned)time(0));  // 난수 발생을 위한 초기화
   hdc = GetDC(hwnd);

 SelectObject(hdc, CreateSolidBrush(RGB(rand()%256, rand()%256, rand()%256)));
      // 스레드마다 사각형 그릴때 사용할 자신의 색을 선택함(난수 이용)

   for(i=0; i<=10; i++)  // 스레드마다 10번 난수 발생
   {
      int num;
      num = rand()%500;   // 스레드마다 자신의 숫자를 0~499 사이에서 선택
      Sleep(3000);
      Rectangle(hdc, 0, 0, 20, num);  // 선택한 숫자 크기의 사각형을 (0,0) 기준으로 그림
   }
   ReleaseDC(hwnd, hdc);
   return;
}
```

10-1 멀티 스레드 기초

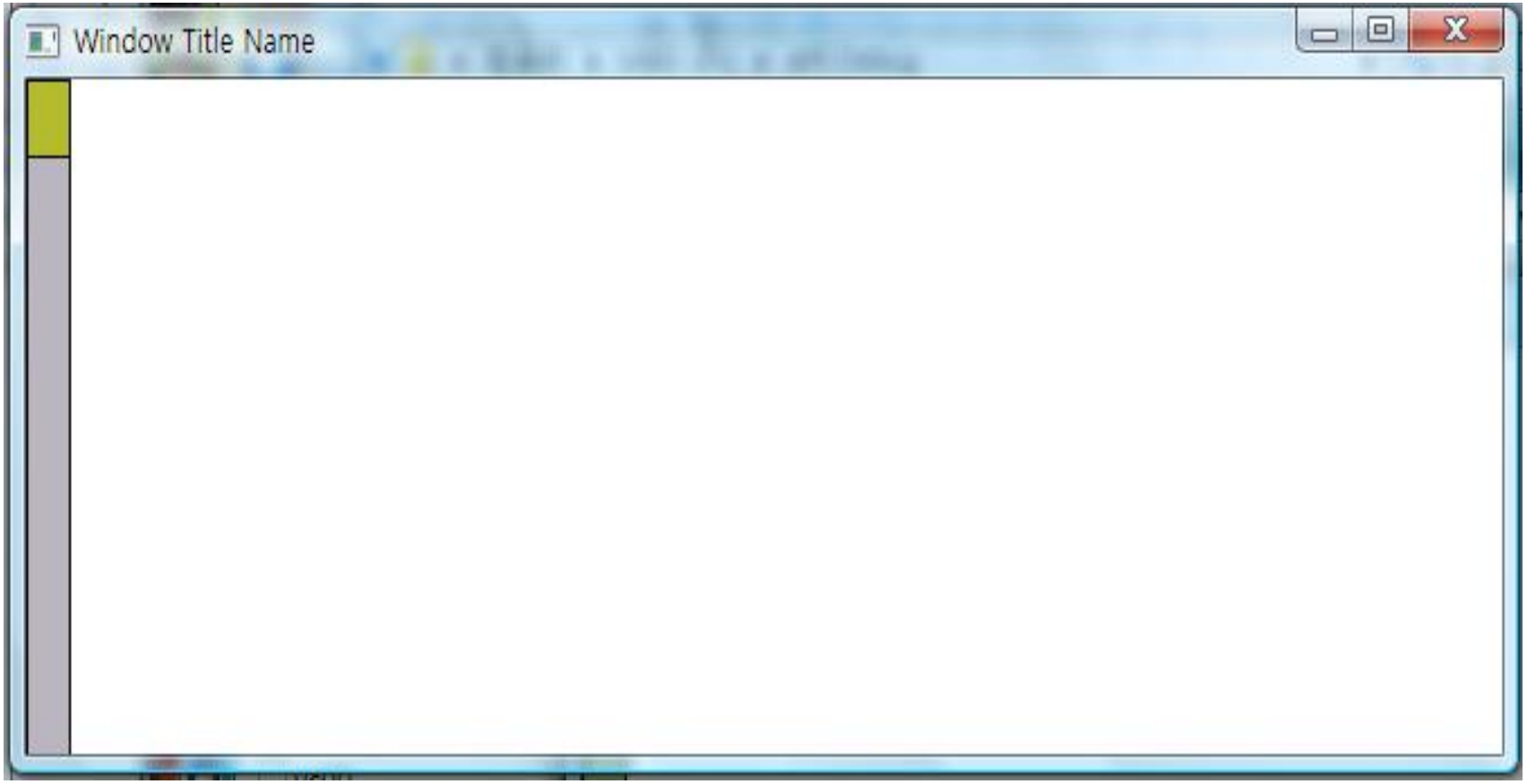
```
#define THREAD_NUM 10
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  static HANDLE hThread[THREAD_NUM];
  static int count;
  int i;
  switch (iMsg)
  {
    case WM_LBUTTONDOWN:
      for (i=0; i<THREAD_NUM; i++)  // 10개의 스레드를 2초 간격을 두고 수행시킴
      {
        hThread[count] = (HANDLE)_beginthreadex (
                                                NULL, 0,
                                                (unsigned int(__stdcall *)(void *))ThreadProc,
                                                NULL, 0, NULL);
                                                // 스레드 함수에게 전달하는 인수는 없음
        Sleep(2000);
      }
      break;
```

10-1 멀티 스레드 기초

```
    case WM_DESTROY:
        for (i=0; i<THREAD_NUM; i++)
          CloseHandle(hThread[i]);
        PostQuitMessage (0) ;
        return 0;
  }
  return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

10-1 멀티 스레드 기초

- 모든 스레드가 동일한 위치에 출력함으로 스레드간 구분 불가



스레드 함수에 인수 전달

- 실습 10-1 문제
  - 모든 스레드가 동일한 위치에 자신의 사각형을 출력하고 있기 때문에 각 스레드에서 만들어내는 결과를 구분하기 어렵다.

- 해결방법
  - 각 스레드가 출력할 위치를 파라미터로 스레드 함수에게 전달

```
hThread[i] = (HANDLE)_beginthreadex(
                        NULL,
                        0,
                        (unsigned int(__stdcall *)(void *))ThreadProc,
                        (void *)&xPos[i],
                            // 출력위치값이 저장된 공간의 주소를  전달
                        0,
                        NULL);
```

10-2 스레드 함수에 인수 전달

. . . 생략 . . .

```
void ThreadProc (void *arg) {
   HDC hdc;
   int i, xPos = *((int *)arg); // 받은 값을 지역변수에 복사, 형변환 주의
   srand((unsigned)time(0));
   hdc = GetDC(hwnd);
   SelectObject(hdc, CreateSolidBrush(
               RGB(rand()%256, rand()%256, rand()%256)));
   for(i=0; i<=10; i++) {
      int num;
      num = rand()%500;
      Sleep(3000);
      Rectangle(hdc, xPos, 0, xPos+20, num);  // 각 스레드마다 출력 위치 다름
   }
   ReleaseDC(hwnd, hdc);
   return;
}
```

10-2 스레드 함수에 인수 전달

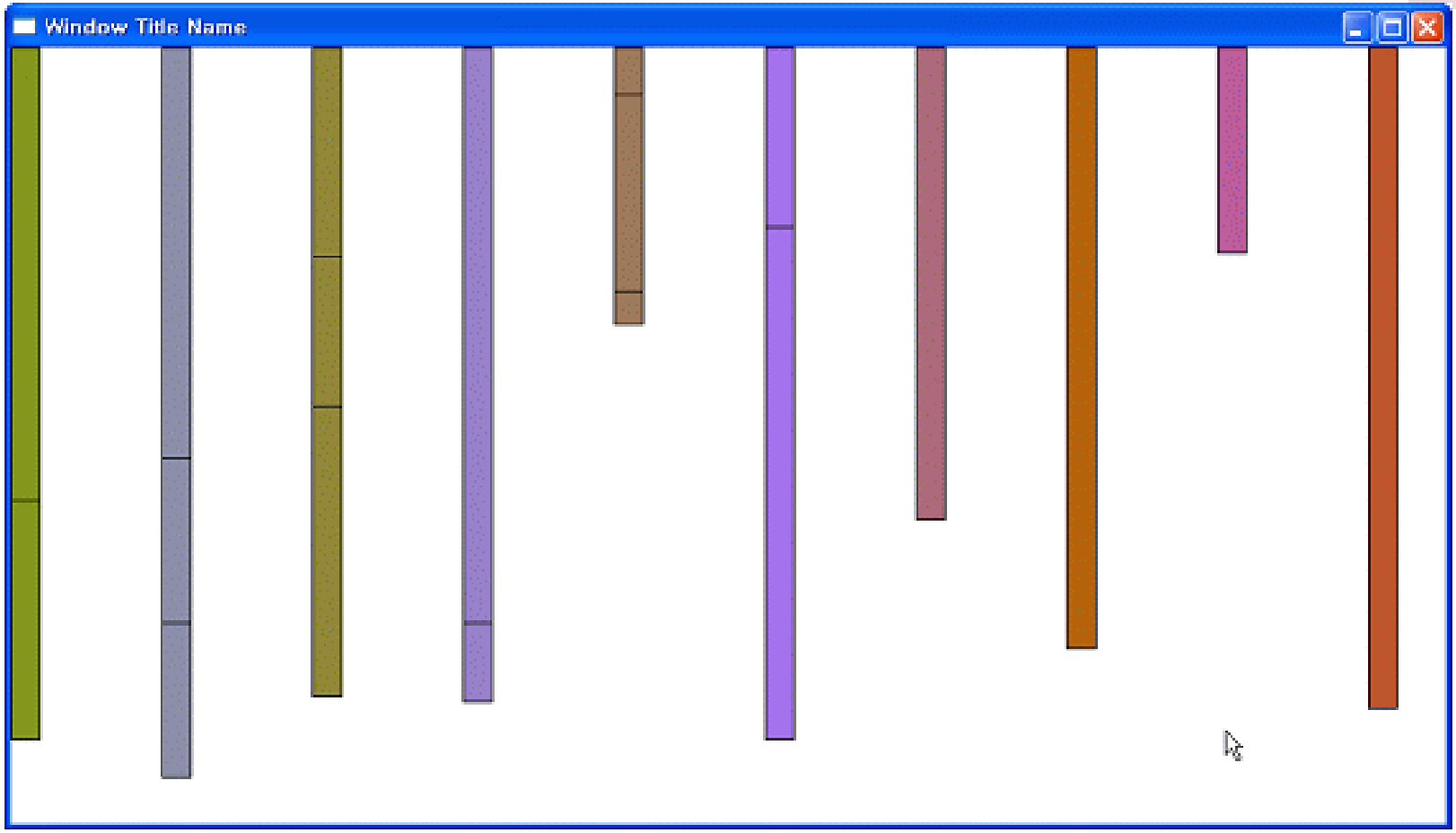
```
#define THREAD_NUM 10

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM
    wParam, LPARAM lParam)
{
   static HANDLE hThread[THREAD_NUM];
   static int xPos[THREAD_NUM]; // 각 스레드가 출력할 위치를 저장할 변수
   int i;
   switch (iMsg)
   {
      case WM_CREATE:
           for (i=0; i<THREAD_NUM; i++)
              xPos[i] = i*100;  // 출력할 위치를 저장
           break;
```

10-2 스레드 함수에 인수 전달

```
case WM_LBUTTONDOWN:
        for (i=0; i<THREAD_NUM; i++)
        {
          hThread[i] = (HANDLE) _beginthreadex (
              NULL,
              0,
              (unsigned int(__stdcall *)(void *))ThreadProc,
              (void *)&xPos[i], // 값이 저장된 공간의 주소를 넘김
               0,
               NULL);
          Sleep(2000);
        }
        break;
. . . 생략 . . .
```

10-2 스레드 함수에 인수 전달

- 10개의 스레드가 서로 다른 공간에 자신의 결과를 출력
- 각 스레드가 발생시키는 난수를 확인 가능

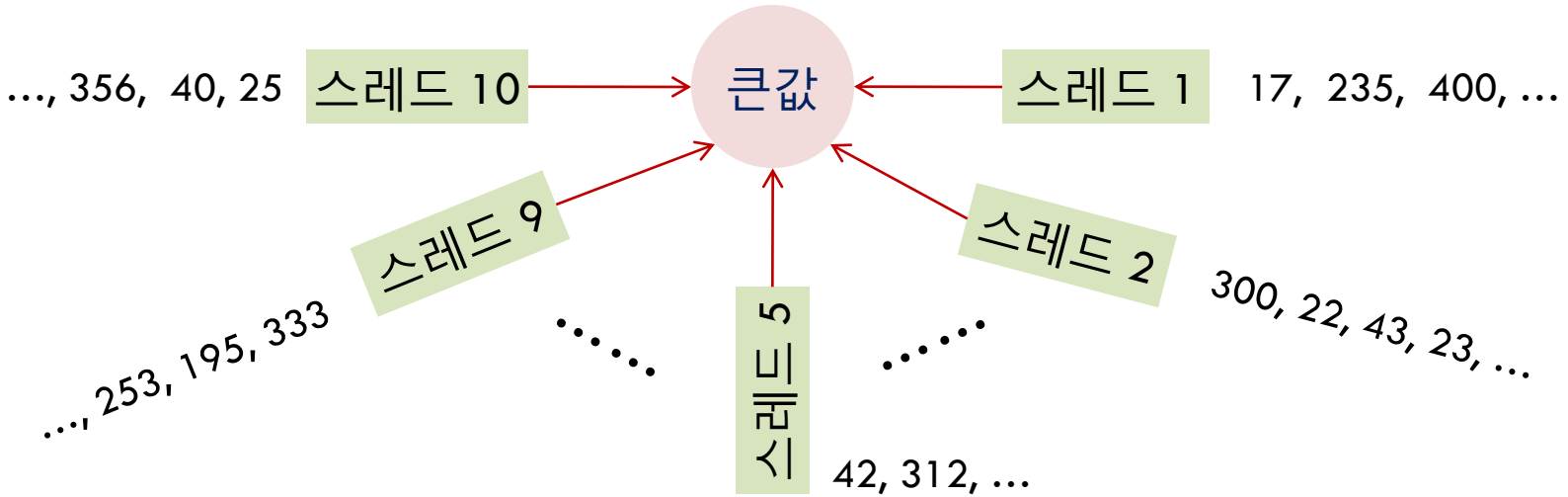


2절. 스레드 동기화

- 스레드간 공간 공유
  - 스레드들이 하나의 문제를 해결하기 위해 협력해야 한다면 동일한 공간을 공유하여 작업을 해야 함
  - 공유하는 동일한 공간을 서로 수정한다고 하면 동기화 문제를 해결해야 함

동기화 필요성

- 실습문제의 변형
  - 10-2에서는 모든 스레드가 독립적으로 수행
  - 각 스레드가 만들어내는 값들 중 가장 큰 값을 전역변수에 저장
  - 각 스레드는 자신이 만든 수를 그 전역변수의 내용과 비교하여 자신이 만든 수가 더 크면 전역변수에 저장



10-3 전역변수에 대한 영향

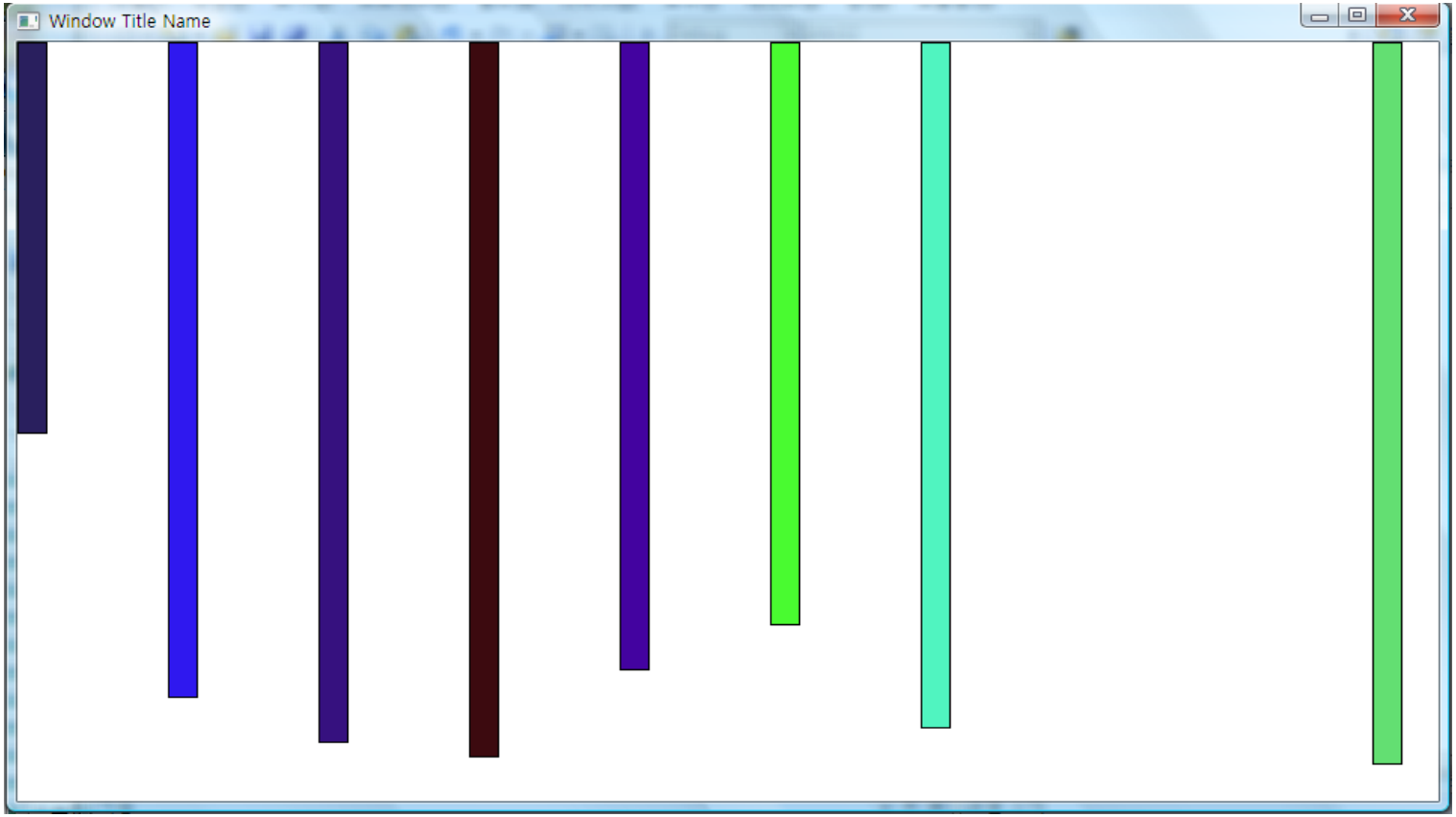
```
int max; // 최대값 저장할 전역변수
void ThreadProc(void *arg)
{
  . . . 생략 . . .
  for(i=0; i<=10; i++)
  {
    int num;
    num = rand()%500;
    if (num > max)                                  // 최대값과 스레드에서 만든 수를 비교
    {
          Sleep(3000);                              // 스레드가 꼭 해야 하는 어떤 일을 가정
          max = num;                                // 최대값에 자신이 만든수를 저장
          Rectangle(hdc, xPos, 0, xPos+20, num);// 이전 수보다 큰 수 일 때만 사각형을 그림
    }
  }
  ReleaseDC(hwnd, hdc);
  return;
}
```

10-3 전역변수에 대한 영향

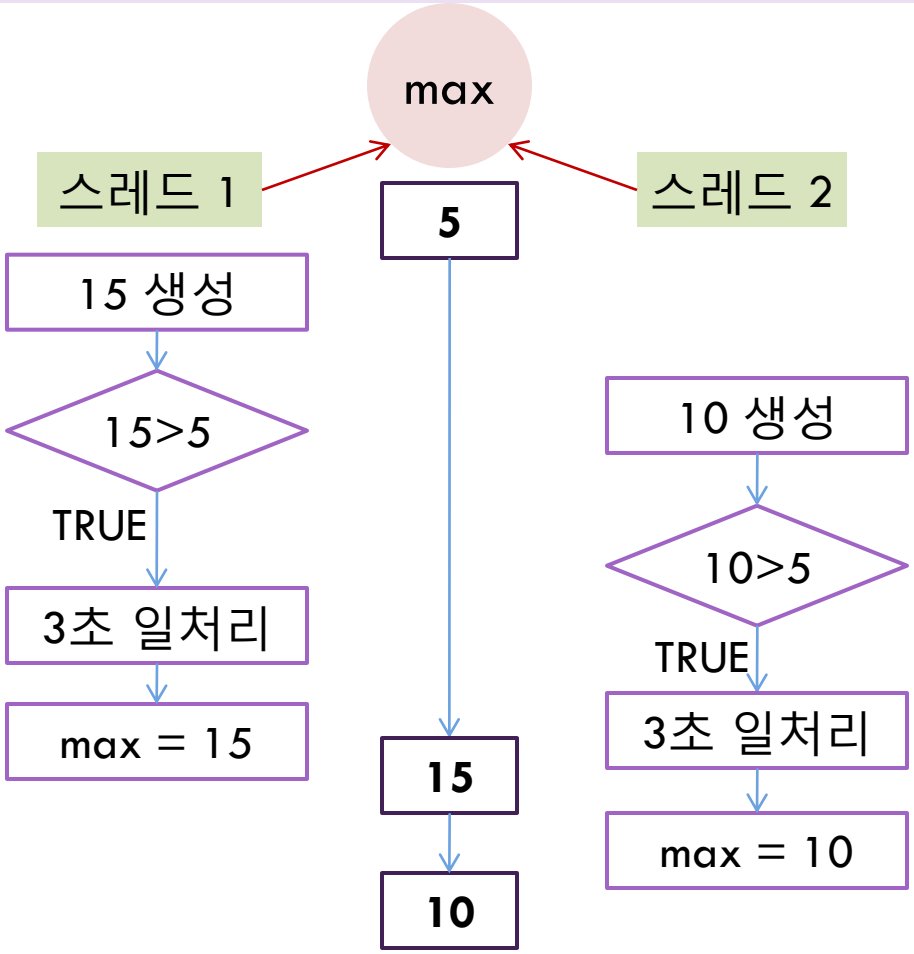
```
#define THREAD_NUM 10
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  . . . 생략 . . .
  switch (iMsg)
  {
    case WM_CREATE:
        . . . 생략 . . .
        max = 0;
        break;
    case WM_LBUTTONDOWN:
        for (i=0; i<THREAD_NUM; i++)
        {
          hThread[i] = (HANDLE)_beginthreadex(
              . . . 생략 . . .
          Sleep(2000);
        }
        break;
```

10-3 전역변수에 대한 영향

- 실행 결과
  - 단일 스레드가 그리는 사각형은 점점 더 커짐
  - 다른 스레드가 그린 사각형보다 작은 사각형 그릴 수 있음 =>문제점



10-3의 문제점


max
스레드 1
스레드 2
5
15 생성
15>5
TRUE
3초 일처리
max = 15
15
10
10 생성
10>5
TRUE
3초 일처리
max = 10

동기화

- 스레드간 동기화
- 스레드 동기화를 위한 방법
  - 크리티컬섹션
  - 세마포어
  - 뮤텍스
  - 이벤트: 여기서 사용
    - 스레드간의 작업 순서나 시기를 조정하고 신호를 보내기 위해 사용
    - 특정한 조건이 만족될 대가지 대기해야 하는 스레드가 있을 경우 이 스레드의 실행을 이벤트로 제어
    - 원하는 작업이 끝났을 때 이벤트를 보내 관련된 다른 작업을 하도록 지시
    - 스레드간에 이벤트 동기호 객체를 사용한다.
    - 자동 리셋 이벤트: 대기 상태가 종료되면 자동으로 비신호 상태가 된다.
    - 수동 리셋 이벤트: 스레드가 비신호상태로 만들 때까지 신호 상태를 유지한다.

이벤트 관련 함수

이벤트 생성 함수

```
HANDLE CreateEvent (     // 이벤트 핸들값 반환
   LPSECURITY_ATTRIBUTES lpEventAttributes,
         // 보안속성, 보통 NULL 이용
   BOOL bManualReset,
         // 시그널, 넌시그널 변환을 수동으로 할것인지 여부
         // TRUE이면 수동 리셋 이벤트
   BOOL bInitialState, // 초기 상태
         // TRUE: 이벤트를 생성하면 시그널 상태로 만듬
         // FALSE: 이벤트를 생성하면 넌시그널 상태로 만듬
   LPCTSTR lpName // 이벤트 이름, 보통 NULL
);
```

이벤트 세팅 함수

```
BOOL SetEvent ( // 이벤트를 시그널 상태로 변환
   HANDLE hEvent // 이벤트 핸들
);
```

이벤트 관련 함수

- 이벤트 리셋 함수

```
BOOL ResetEvent(                    // 이벤트를 넌시그널 상태로 변환
    HANDLE hEvent                   // 이벤트 핸들
    );
```

- 단일 객체를 대기하는 함수

```
DWORD WaitForSingleObject(
    HANDLE hHandle,                 // 대기할 객체의 핸들
    DWORD dwMilliseconds            // 대기 시간,
                                    // INFINITE는 무한정 기다린다는 뜻임
);
```

이벤트 관련 함수

- 복수 객체를 대기하는 함수

```
DWORD WaitForMultipleObjects(
   DWORD nCount,                    // 기다릴 객체의 개수
   const HANDLE *lpHandles,         // 기다릴 객체들의 핸들들
   BOOL bWaitAll,
      // TRUE: 모든 객체가 시그널 상태가 되는 것을 기다림
      // FALSE: 1개의 객체라도 시그널 상태가 되는 것을 기다림
   DWORD dwMilliseconds             // 대기 시간
);
```

10-4 이벤트 실습

```
. . . 생략 . . .
void ThreadProc(void *arg) {
   . . . 생략 . . .
   for(i=0; i<=10; i++) {
      int num;
      WaitForSingleObject(hEvent, INFINITE);
            // hEvent가 시그널 상태가 될때까지 무한정 기다림
            // 시그널 상태가 되면 대기중인 스레드중 하나만 진행
            // hEvent는 다시 넌시그널 상태로 됨
      num = rand()%500;
      if (num > max) {
         Sleep(3000);
         max = num;
         Rectangle(hdc, *pX, 0, *pX+20, num);
      }
      SetEvent(hEvent);    // hEvent를 시그널 상태로 변환
   }
   . . . 생략 . . .
}
```

10-4 이벤트 실습

```
. . . 생략 . . .
  case WM_LBUTTONDOWN:
     hEvent = CreateEvent(NULL,FALSE,TRUE,NULL);
          // hEvent를 생성하고 자동변환에 시그널 상태로 셋팅
     for (i=0; i<THREAD_NUM; i++)
     {
       hThread[i] = (HANDLE)_beginthreadex(...)
       Sleep(2000);
     }
     WaitForMultipleObjects(THREAD_NUM, hThread,
                            TRUE, INFINITE);
          // 모든 스레드가 종료상태가 될때까지 기다림
     CloseHandle(hEvent);
     break;
. . . 생략 . . .
```

10-4 이벤트 실습

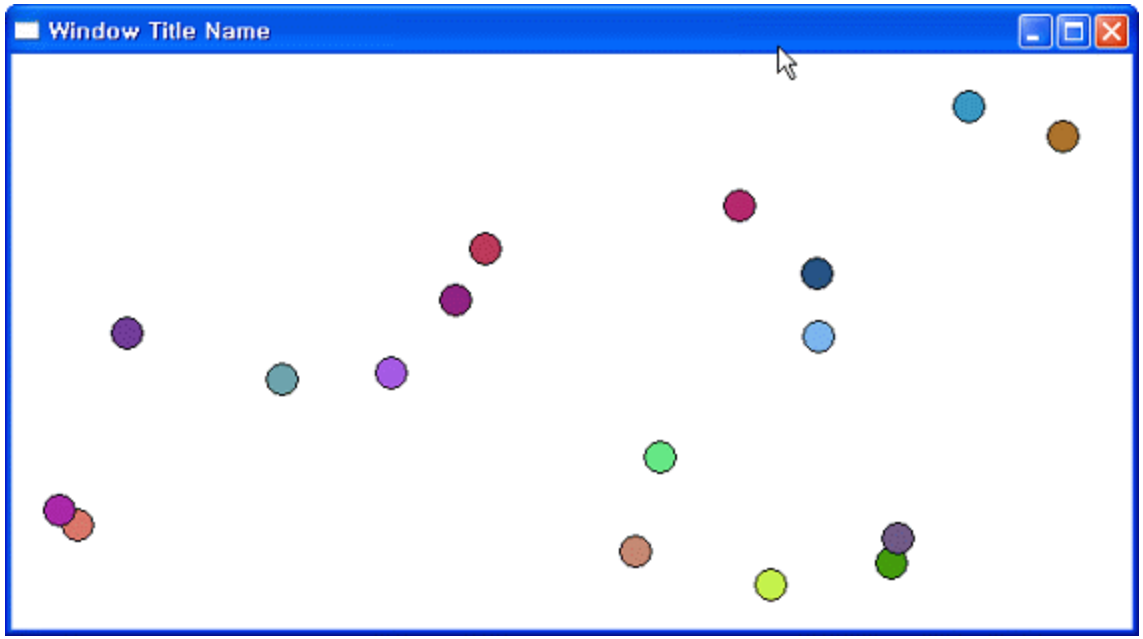
- 실행 결과
  - 제일 큰 수만 max에 저장함
  - 새롭게 등장하는 사각형은 기존의 사각형들보다 항상 큼

실습 10-1

- 제목

  스레드로 화면에 원 그리기

- 내용
  - 마우스 클릭할 때마다 하나의 스레드가 구동됨
  - 스레드는 화면에 랜덤 위치에 하나의 원을 그리고 원의 색을 1초마다 변경, 각 스레드는 10번 색을 변경



실습 10-2

2013년도 1학기

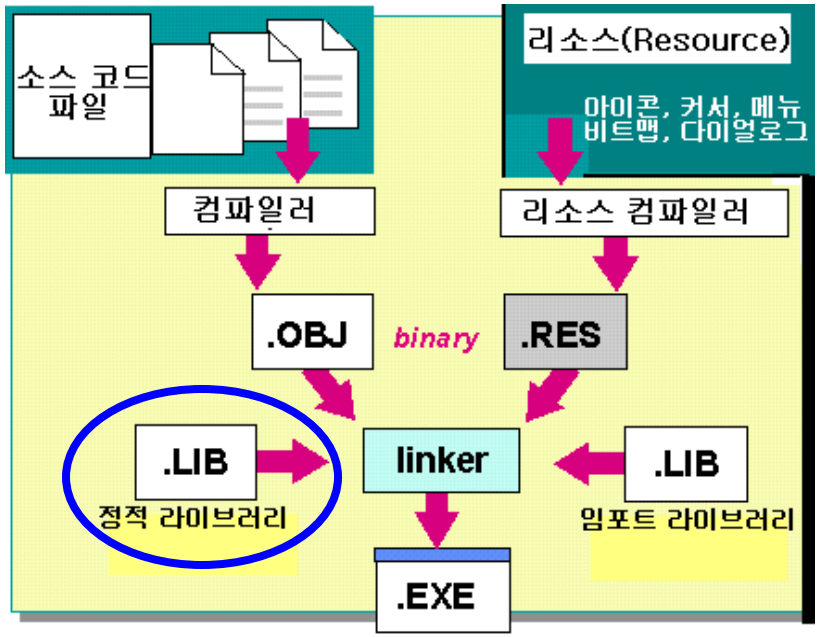
DLL(Dynamic Link Library) 만들기

학습 내용

- DLL 개요
- DLL 만들기
- DLL 사용하기
- 실습문제

정적 링크(static link) 라이브러리

- printf()같은 표준 C 라이브러리 함수는 프로그래머를 위해서 제공하는 함수이다.
- 이러한 함수들은 전부 컴파일되어 .lib파일 형태로 존재한다.
- 이 라이브러리 파일은 소스 프로그램 컴파일 후 링커에 의해 함께 링크되어 실행파일에 포함된다.
- 이러한 라이브러리 연결 방법을 정적 링크(static link)라고 한다.
- 정적 링크 라이브러리는 컴파일시 링크가 되며 실행파일의 일부로 포함된다.
- 표준 함수뿐만 아니라 자신이 많이 사용하는 모듈을 .lib파일로 만든 후 현재 프로젝트에 포함시키고 컴파일/링킹을 하면 언제든지 관련 모듈은 따로 프로그래밍할 필요가 없다



- 정적 라이브러리를 사용하는 프로그램은 이미 실행 파일에 관련 모듈이 포함되어 있으므로 실행할 때는 해당 .lib파일 없이 실행할 수 있는 장점을 갖는다.
- 그러나 .lib파일이 실행 파일의 일부가 되므로 실행파일의 크기가 대체로 커진다.

동적 링크 라이브러리 (DLL: Dynamic Link Library)

- DLL은 컴파일시가 아닌 실행시에 관련 모듈을 링크
  - 링크시 실행파일에 포함되지 않는다.
  - 링크 시에는 라이브러리 이름과 라이브러리에서 사용한 함수의 이름만 실행파일에 기록된다.
  - 실제로 실행파일이 실행되면 그때 라이브러리가 메모리에 따로 로드되고 라이브러리의 함수를 호출하게 되면 그 함수로 실행이 넘어가게 된다.

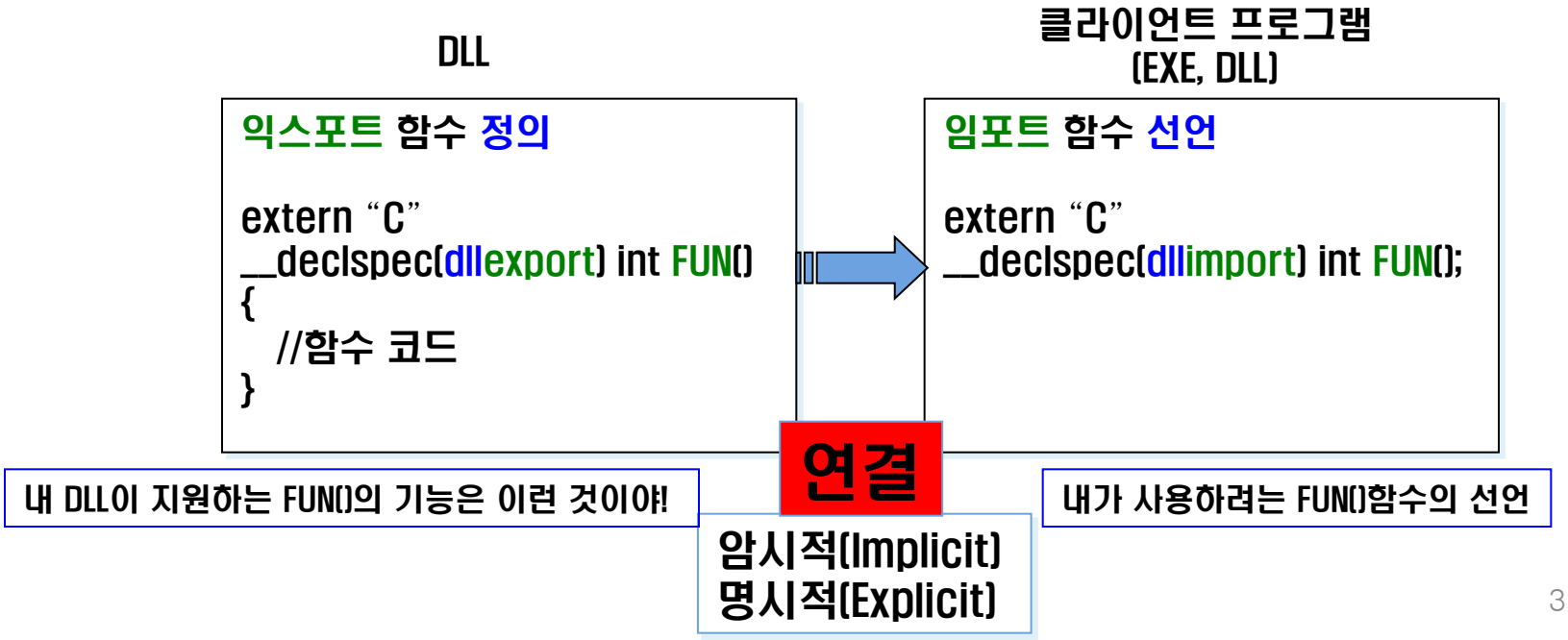
- DLL을 사용했을 때,
  - 일반적으로 DLL을 사용하는 프로그램은 실행파일 크기가 작다.
  - DLL을 사용하는 프로그램은 DLL만 교체하면 프로그램의 버전업을 쉽게 할 수 있다.

DLL의 특징

- 정적 라이브러리는 소스만 제공할 수 있지만 동적 라이브러리는 리소스도 제공할 수 있는 장점이 있다.
  - 자주 사용하는 비트맵 리소스가 있다면 이를 DLL파일에 저장하여 필요할 때 연결해 쓸 수 있다.
  - 대표적으로 글꼴 파일(.fon, .ttf)이 리소스만 갖는 DLL이다.
- DLL은 확장자가 반드시 .dll일 필요는 없지만 확장자가 .dll인 DLL만이 윈도우즈에 의해 자동적으로 로드된다.
  - DLL이 다른 확장자를 가지면, LoadLibrary()나 LoadLibraryEx()함수를 사용하여 강제로 로드한다.
- DLL을 사용할 경우 실행파일의 크기는 작아지지만 DLL파일을 별도로 배포해야 하며, 프로그래밍하기 복잡하고 버전관리 등에 신경을 써야 한다.
- 디바이스 드라이버(keyboard.drv mouse.drv 등)나 각종 커스텀 컨트롤들은 모두 DLL로 만들어진다.
- 공통 컨트롤은 comctl32.dll로 제공되며, 공통 대화상자는 comdlg32.dll로 제공된다.

DLL을 만들고 사용할 때의 규칙

- 익스포트(export)
  - 함수를 제공하는 DLL에서는 자신이 제공하는 함수들의 정보를 공개해야 한다.
- 임포트(import)
  - 만들어진 DLL를 사용하는 클라이언트(EXE나 DLL)에서는 어떤 DLL에 있는 어떤 함수를 사용한다는 선언을 해야 한다.
- 클라이언트에서 DLL의 함수를 호출하는 방법에는 암시적(implicit)방법과 명시적(explicit)방법이 있는데 다음 그림은 암시적 방법을 나타낸다.



DLL을 만들고 사용할 때의 규칙

- DLL에서 함수를 익스포트하거나 클라이언트에서 임포트할 때는
  - __declspec과 extern”C” 를 같이 써야 한다.

  - DLL에서 익스포트되는 함수 선언
    extern “C” __declspec(dllexport)
  - 클라이언트에서는 임포트하는 함수의 원형을 사용
    extern “C” __declspec(dllimport)

DLL 만들기(dll1.dll)

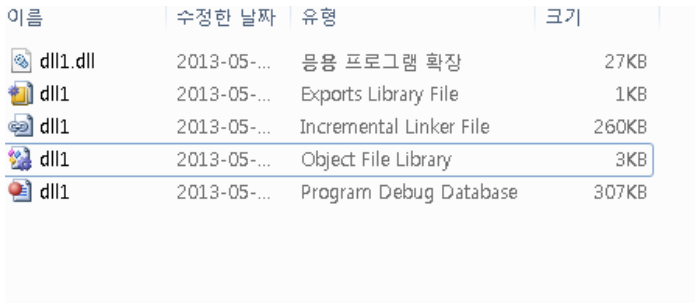
- Visual C++에서 Win32 API로 DLL을 만들 때는 응용 프로그램에서 DLL을 선택
- Project name: dll1
- 디폴트로 설정되어 있는 "An empty DLL project"
- File/New/C++ Source File로 옆에 있는 사칙연산을 하는 4개의 함수를 갖는 cpp파일을 추가

- 디버그 모드로 컴파일하면 Debug폴더에 dll1.dll과 dll1.lib파일이 생긴다.

- dll1.lib파일을 임포트 라이브러리(import library)라고 한다.
- 이 파일은 DLL에 익스포트되는 함수 정보와 그 함수 코드가 어떤 DLL에 정의되어 있는지에 대한 정보를 가진 파일로 앞에서 설명한 일반 정적 lib파일과는 다르다.

- Visual C++은 DLL을 컴파일할 때 .dll과 같은 이름의 임포트 라이브러리(.lib)를 자동으로 만들어 준다.

```
extern "C"
__declspec(dllexport) int Add(int x, int y)
{
   return x+y;
}
extern "C"
__declspec(dllexport) int Sub(int x, int y)
{
   return x-y;
}
extern "C"
__declspec(dllexport) int Mul(int x, int y)
{
   return x*y;
}
extern "C"
__declspec(dllexport) double Div(int x, int y)
{
   return (double)x/y;
}
```



```
extern "C"
__declspec(dllexport) int Add(int x, int y)
{    return x+y;   }
```

네임 맹글링(name mangling)

- 네임 데코레이션(name decoration)이라고도 한다.
  - 컴파일러가 호출 규약과 C/C++ 문법에 따라 컴파일할 때 함수명에 약간의 변형을 가하는 것을 말한다.
  - 소스 코드에 사용된 함수명과 컴파일된 후의 이진코드에서 링크 정보를 제공하기 위해 사용되는 함수명이 다르다.
- C언어와 달리 C++에서는 함수의 중첩(overloading)이 가능하다.
  - 이는 매개변수의 개수나 형이 다른 유사한 기능의 함수를 하나의 이름으로 여러 개 구현할 수 있다는 것이다.
  - 컴파일러는 컴파일할 때 이들 같은 이름의 함수들을 구분하기 위해 함수명 이외에 인자형을 붙여서 새로운 함수이름을 만든다.
- *extern "C"는 C++로 컴파일할 때 사용하는 이러한 함수명 변경을 하지 않도록 하는 것이다.*
- 즉, C++함수를 C언어 형식의 함수로 만들어서 다른 언어에서 호출할 수 있도록 한다.

C++의 함수 중첩(overloading)

```
#include <iostream.h>
int add(int i, int j)
{
    return (i+j);
}

double add(double i, double j)
{
    return (i+j);
}

void main()
{
    cout<<add(10,20)<<endl;
    cout<<add(10.5,20.3)<<endl;
}
```

한 프로그램에 이름이 같은 함수 여러 개 사용가능

```
extern "C"
__declspec(dllexport) int Add(int x, int y)
{    return x+y;   }
```

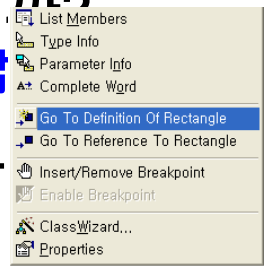
extern "C"

- 다음과 같은 헤더를 갖는 두 개의 함수를 함수중첩으로 한 프로그램에 작성하고 C++로 컴파일하면 함수명은 각각 다음과 같이 바뀐다.
  - int __stdcall Add(int a, int b, int c, int d)
    - ☞ ?Add@@YGHHHHH@Z
  - int __stdcall Add(int a, int b, int c, char d)
    - ☞ ?Add@@YGHHHHD@Z
- 원래 함수명은 Add이지만 "?Add@@YGHHHHH@Z"로 데코레이션되었다.
- 여기서 Add라는 함수명 외에 추가된 문자들은 매개변수의 개수나 형에 대한 정보이다.
- 그러나 앞에 extern "C"를 붙이고 컴파일하면 함수명은 다음과 같다.
  - extern "C" int __stdcall Add(int a, int b, int c)
    - ☞ _Add@12
- extern "C"를 사용하면 함수명 앞에 "_"(underscore) 문자가 붙고 함수명 다음에 "@"문자가 온 후 매개변수들의 byte 크기를 가리키는 숫자(10진수)가 저장된다.
- C문법으로 컴파일되었으므로 단지 함수명 앞에 "_" 문자만 추가되었으며 데코레이션이 되지 않는다.

```
extern "C"
__declspec(dllexport) int Add(int x, int y)
{    return x+y;   }
```

__declspec(dllimport)

- 개발자 작업장에서 Rectangle()함수에 캐럿을 위치시키고 마우스 오른쪽 버튼을 누르면 나오는 팝업 메뉴에서 "Go To Definition Of Rectangle"이라는 메뉴를 선택해보자.
- 그러면 wingdi.h파일을 열리며 Rectangle함수가 다음과 같은 원형을 가지고 있음을 알 수 있다.
  - **WINGDIAPI BOOL WINAPI Rectangle(HDC, int, int, int, int);**
- 모든 전처리기들을 전개하여 Rectangle()함수의 원형을 다시 쓰면 다음과 같다.
  - **__declspec(dllimport) BOOL WINAPI Rectangle(HDC, int, int, int, int);**
- WINAPI는 __stdcall방식의 함수 호출 규약, BOOL은 이 함수의 리턴형이다.
- 그러면 제일 앞에 있는 __declspec(dllimport)은 무엇일까?
  - 이것은 현재 프로그램에서 DLL(gdi32.dll)의 rectangle()함 사용할 것이라는 것을 나타낸다.
  - GDI관련 Win32 API함수는 모두 이 gdi32.dll파일이 제공한



```
extern "C"
__declspec(dllexport) int Add(int x, int y)
{    return x+y;   }
```

__declspec

- extern "C"다음에 나오는 __declspec는 기억 클래스(storage-class)정보를 단순화, 표준화하기 위해 마이크로소프트사에서 C/C++문법을 확장한 확장 속성 구문(extended attribute syntax)이다.

- dllimport는 DLL에 있는 데이터, 객체, 함수를 임포트한다.
  - DLL에 있는 함수를 사용하겠다는 선언이다.
  - DLL을 사용하는 클라이언트(EXE나 DLL)에서는 어떤 DLL에 있는 어떤 함수를 사용한다는 선언을 해야 하는데 이것을 임포트한다고 한다.

- dllexport는 DLL에 있는 데이터, 객체, 함수를 익스포트한다.
  - 함수를 제공하는 DLL에서는 자신이 제공하는 함수들의 정보를 공개해야 한다.
  - 그래야 그 함수를 사용할 수 있기 때문이다.

extern "C" 와 __declspec(dllexport)

```
extern "C"
__declspec(dllexport) int Add(int x, int y)
{
    return x+y;
}
```

함수이름 변경하지 않게
C문법으로 컴파일해야한다.

함수를 제공하는 DLL에서는 자신이 제공하는 함수들의 정보를 공개해야 한다.

DLL 사용하기

- 만든 DLL을 사용하기 위해서는 사용하는 클라이언트(EXE나 DLL)프로그램이 만들어진 DLL을 연결해야 한다.

- 암시적(implicit) 또는 묵시적 방법
  - .dll파일과 함께 제공되는 임포트 라이브러리(.lib)로 연결하는 방법이다.
  - 프로그램이 실행되면 DLL도 함께 로드되며 프로그램이 종료되면 DLL도 함께 해제된다.

- 명시적(explicit) 방법
  - LoadLibrary(), GetProcAddress(), FreeLibrary()같은 Win32 API함수를 사용하는 방법
  - 암시적 방법에 비해 장점이 있으나 함수 포인터 같은 어려운 개념을 알고 있어야 한다.

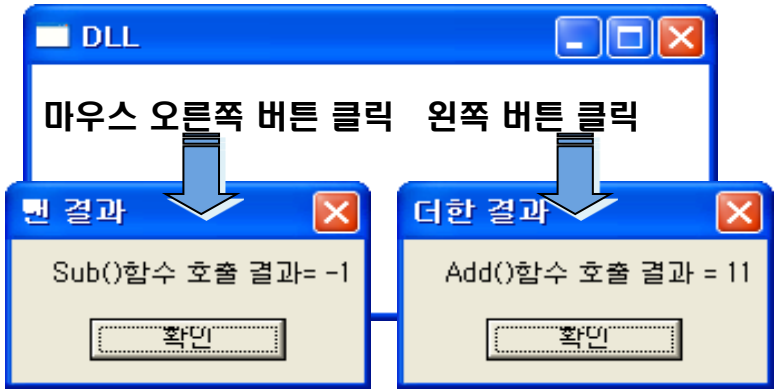
DLL 연결

	암시적(implicit)방법	명시적(explicit)방법
방 법	임포트 라이브러리(.lib)사용	API함수 사용
DLL이 메모리에 로드되는 시점	프로그램이 실행되면 DLL도 함께 로드됨	LoadLibrary()함수 호출시
DLL이 메모리에 해제되는 시점	프로그램이 종료되면 DLL도 함께 해제	FreeLibrary()함수 호출시
익스포트 함수 사용 시점	프로그램이 실행되는 동안 어디서나	GetProcAddress()함수 호출 후
장 점	함수 호출이 간편	메모리 소비 적고, DLL로딩 실패시에도 대응코드 가능
단 점	메모리 낭비와, DLL로딩 실패시 프로그램 실행 불가	함수 호출이 복잡
사용 예	1) .lib와 .dll을 현재 프로젝트 폴더에 복사 2)Project/Setting/Link탭을 열어 임포트 라이브러리 포함시킴 3)임포트 함수 선언 4)함수 사용	1)LoadLibrary() 2)GetProcAddress() 3)함수 사용 4)FreeLibrary()

암시적 방법으로 dll사용

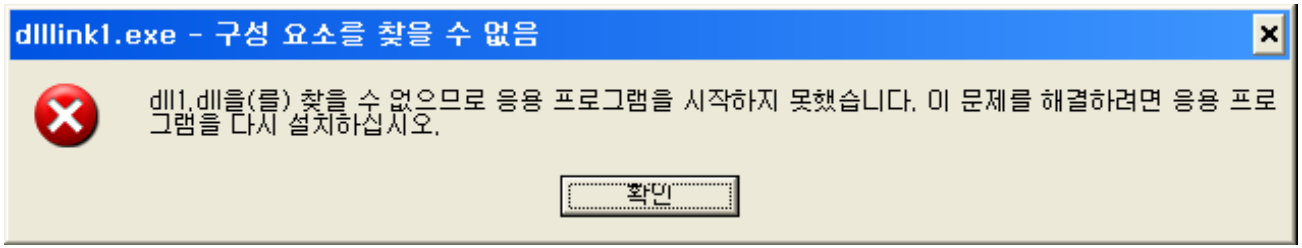
- 임포트 라이브러리를 사용하는 암시적(implicit) 방법으로 위에서 만들어 놓았던 사칙 연산함수를 갖는 dll1.dll을 사용하는 프로그램을 작성하시오.

- 이 프로그램은 마우스 오른쪽 버튼을 누르면 Sub(5,6)한 결과를 메시지 박스로 출력하고, 마우스 왼쪽 버튼을 누르면 Add(5,6)한 결과를 메시지 박스로 출력한다.



암시적 방법

- 앞에서 만들어 놓았던 dll1.lib와 dll1.dll을 현재 프로젝트 폴더에 복사한다.
- 이 프로그램을 실행할 때 반드시 다음의 디렉토리 중 하나에는 dll1.dll파일이 있어야 하며 윈도우즈는 다음과 같은 순서로 DLL을 찾는다.
  - 실행 파일이 로드된 디렉토리
  - 현재 디렉토리
  - 윈도우즈 시스템 디렉토리(C:\Windows\System32)
  - 윈도우즈 디렉토리(C:\Windows)
  - PATH환경변수로 지정된 디렉토리
- 만약 위의 순서대로 찾아서 해당 DLL이 없으면 이 프로그램을 실행할 때 다음과 같은 에러 메시지가 출력된다.



암시적 방법(계속)

- 하나의 "Win32 Application"프로젝트를 생성하고 다음 페이지 소스를 cpp파일로 프로젝트에 포함시킨다.
- “프로젝트/속성” 메뉴 -> 링커/입력 -> 추가종속성에 임포트 라이브러리(*.lib)를 포함시킨다.

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
HINSTANCE g_hInst;

extern "C" __declspec(dllimport) int Add(int, int);
extern "C" __declspec(dllimport) int Sub(int, int);
//임포트함수 선언
int APIENTRY WinMain( HINSTANCE hInstance,HINSTANCE
   hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
{
 static char szClassName[]  = "ddd";
 static char szTitle[] = "DLL";

 MSG       msg;
 HWND      hWnd;
 WNDCLASSEX   wc;
 g_hInst=hInstance;

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

 hWnd = CreateWindow(
          szClassName, szTitle, WS_OVERLAPPEDWINDOW,
         CW_USEDEFAULT,  CW_USEDEFAULT,
         CW_USEDEFAULT,  CW_USEDEFAULT,
         NULL, NULL, hInstance, NULL );
 if ( !hWnd )    return( FALSE );
 ShowWindow(hWnd, nCmdShow);  UpdateWindow(hWnd);

 while( GetMessage( &msg, NULL, 0, 0) )
 {   TranslateMessage( &msg );
   DispatchMessage( &msg ); }
 return msg.wParam;
}
```

```
extern "C"
__declspec(dllexport) int Add(int x, int y)
{
  return x+y;
}
extern "C"
__declspec(dllexport) int Sub(int x, int y)
{
  return x-y;
}
```

dll1.dll의 내용

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg,
                         WPARAM wParam, LPARAM lParam)
{
  char add[50];
  char sub[50];

  switch ( uMsg ){
  case WM_LBUTTONDOWN:
   wsprintf(add, "Add()함수 호출 결과 = %d",Add(5,6));
   MessageBox(hWnd,add,"더한 결과",MB_OK);
  break;

  case WM_RBUTTONDOWN:
   wsprintf(sub, "Sub()함수 호출 결과= %d",Sub(5,6));
   MessageBox(hWnd,sub,"뺀 결과",MB_OK);
  break;

  case WM_DESTROY:
     PostQuitMessage(0);
     break;

  default :
     return DefWindowProc( hWnd, uMsg, wParam, lParam );

  }
  return 0;
}
```

암시적 방법: 소스설명

- 소스에서 중요한 부분은 4, 5번째 줄이다.
  - extern "C" __declspec(dllimport) int Add(int, int);
  - extern "C" __declspec(dllimport) int Sub(int, int);
- 이것은 DLL(dll1.dll)에서 사용할 임포트 함수를 선언하는 것이다.
- dll1.dll은 4개의 함수를 가지고 있지만 여기서는 Add(),Sub()함수만 사용한다는 것이다.
- 이렇게 선언하면 이제부터 이 프로그램에서 이 함수들을 바로 호출할 수 있다.
- 호출한 예는 다음과 같다.
  - wsprintf(add, "Add()함수 호출 결과 = %d",Add(5,6));
  - wsprintf(sub, "Sub()함수 호출 결과= %d",Sub(5,6));
- 마우스 오른쪽 버튼을 누르면 Mul(5,6)한 결과를 메시지 박스로 출력하고, 마우스 왼쪽 버튼을 누르면 Div(5,6)한 결과를 메시지 박스로 출력

암시적 방법

- dll1.dll의 곱셈(Mul) 나눗셈(Div)하는 함수 사용하기

```
extern "C"
__declspec(dllexport) int Mul(int x, int y)
{
   return x*y;
}
extern "C"
__declspec(dllexport) double Div(int x, int y)
{
   return (double)x/y;
}
```

명시적(explicit) 연결 방법

- LoadLibrary(), GetProcAddress(), FreeLibrary()같은 Win32 API함수를 사용한다.
- 사용할 DLL을 메모리에 로드하기 위해 LoadLibrary()함수를 사용하고, FreeLibrary()함수로 사용한 DLL을 메모리에서 해제한다.
- 익스포트 함수를 사용하기 위해서는 GetProcAddress()함수를 호출한다.
- 이 방법은 사용할 시점에만 DLL을 메모리에 로드하므로 메모리 소비가 적다.
- DLL로딩 실패 시에도 대응코드가 가능하다.
- 함수 호출 방법이 복잡하다는 단점을 갖는다.

LoadLibrary(), FreeLibrary(), GetProcAddress()

- 사용할 DLL을 메모리에 로드하는 LoadLibrary()함수

```
HINSTANCE LoadLibrary(
    LPCTSTR lpLibFileName   // address of filename of executable module
);
```

- 익스포트 함수를 사용하기 위한 GetProcAddress()함수

```
FARPROC GetProcAddress(
    HMODULE hModule,    // handle to DLL module
    LPCSTR lpProcName   // name of function
);
```

- 사용한 DLL을 메모리에서 해제하는 FreeLibrary()함수

```
BOOL FreeLibrary(
    HMODULE hLibModule   // handle to loaded library module
);
```

함수 포인터

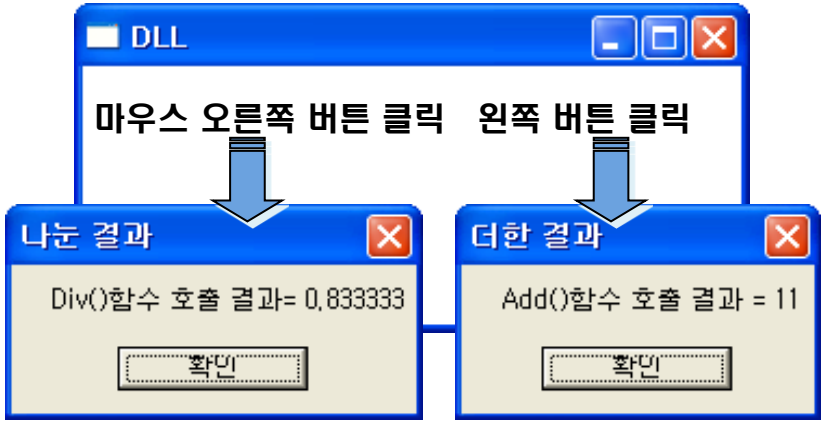
- "int Add(int, int);"라는 함수를 선언했을 경우, Add 함수의 함수 포인터변수 pAdd는 "int(*pAdd)(int, int);"와 같이 선언할 수 있다.
- 함수 포인터 선언은 함수 선언과 유사해 보이지만 이 포인터로는 **함수를 지시하겠다**는 것을 나타낸다.

```
#include <stdio.h>
int Add(int, int);
void main()
{
    int(*pAdd)(int, int);
    pAdd=Add;
    printf("%d",Add(3,4));
    printf("%d",pAdd(3,4));
}
int Add(int x, int y)
{
    return x+y;
}
```

7이 두 번 출력된다.
main()함수에서 pAdd라는 포인터 변수를 선언했는데
이 변수는 지금까지 보았던 포인터 변수와는 차이가 있다.
형과 포인터 변수 이름 사이에 우선 순위 괄호가 있으며
변수명 다음에는 괄호와 인자(int, int)도 보인다.
이러한 포인터를 함수 포인터라 하며 이 포인터로는
함수를 지시할 수 있다.
즉 "pAdd=Add"와 같이 함수명을 함수 포인터 변수에
대입하면 원래 함수와 똑같은 함수로 사용할 수 있다.
그래서 Add(3,4)로 호출하나 pAdd(3,4)로 호출하나
결과는 동일하다.

명시적 방법으로 dll사용

- 임포트 라이브러리를 사용하지 않는 명시적 방법으로 위에서 만들어 놓았던 사칙 연산함수를 갖는 dll1.dll을 사용하는 프로그램을 작성하시오.
- 이 프로그램은 마우스 오른쪽 버튼을 누르면 Div(5,6)한 결과를 메시지 박스로 출력하고, 마우스 왼쪽 버튼을 누르면 Add(5,6)한 결과를 메시지 박스로 출력한다.



```
#include <windows.h>
#include <stdio.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
HINSTANCE g_hInst;

int APIENTRY WinMain( HINSTANCE hInstance,HINSTANCE
   hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
{
 static char szClassName[]  = "ddd";
 static char szTitle[] = "DLL";

 MSG       msg;
 HWND      hWnd;
 WNDCLASSEX   wc;
 g_hInst=hInstance;

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

 hWnd = CreateWindow(
          szClassName, szTitle, WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,  CW_USEDEFAULT,
          CW_USEDEFAULT,  CW_USEDEFAULT,
          NULL, NULL, hInstance, NULL );
 if ( !hWnd )    return( FALSE );
 ShowWindow(hWnd, nCmdShow);  UpdateWindow(hWnd);

 while( GetMessage( &msg, NULL, 0, 0) )
 {   TranslateMessage( &msg );
   DispatchMessage( &msg );  }
 return msg.wParam;
}
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg,
                         WPARAM wParam, LPARAM lParam)
{
  char add[50];
  char div[100];
  HINSTANCE hDllInst;
  int(*pAdd)(int, int);
  double(*pDiv)(int, int);

  switch ( uMsg ){
  case WM_LBUTTONDOWN:
    hDllInst=LoadLibrary("dll1.dll");
    pAdd=(int (*)(int, int))GetProcAddress(hDllInst,"Add");
    wsprintf(add, "Add()함수 호출 결과 = %d",pAdd(5,6));
    MessageBox(hWnd,add,"더한 결과",MB_OK);
    FreeLibrary(hDllInst);
  break;

  case WM_RBUTTONDOWN:
   hDllInst=LoadLibrary("dll1.dll");
   pDiv=(double(*)(int, int))GetProcAddress(hDllInst,"Div");
   sprintf(div, "Div()함수 호출 결과= %lf",pDiv(5,6));
   MessageBox(hWnd,div,"나눈 결과",MB_OK);
   FreeLibrary(hDllInst);
  break;

  case WM_DESTROY:
     PostQuitMessage(0);
     break;

  default :
     return DefWindowProc( hWnd, uMsg, wParam, lParam );

  }
  return 0;
}
```

명시적 방법으로 dll을 연결하여 사용하는 프로그램

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
    lParam)
{
   char add[50];
   char div[100];
   HINSTANCE hDllInst;
   int(*pAdd)(int, int);
   double(*pDiv)(int, int);

   switch ( uMsg ){
   case WM_LBUTTONDOWN:
        hDllInst=LoadLibrary("dll1.dll");
        pAdd=(int (*)(int, int))GetProcAddress(hDllInst,"Add");
        wsprintf(add, "Add()함수 호출 결과 = %d",pAdd(5,6)); //pAdd함수 사용
        MessageBox(hWnd,add,"더한 결과",MB_OK);
        FreeLibrary(hDllInst);
   break;

   case WM_RBUTTONDOWN:
        hDllInst=LoadLibrary("dll1.dll");
        pDiv=(double(*)(int, int))GetProcAddress(hDllInst,"Div");
        sprintf(div, "Div()함수 호출 결과= %lf",pDiv(5,6));
        MessageBox(hWnd,div,"나눈 결과",MB_OK);
        FreeLibrary(hDllInst);
   break;
         :
```

사용할 두 개의 함수 포인터를 선언

dll1.dll파일을 메모리에 로드하며 리턴값을 hDllInst변수에 저장

hDllInst에 있는 Add함수의 주소를 얻어 pAdd에 저장

리턴값을 함수 포인터형으로 캐스트연산

실습 11-1

- 비트맵 리소스만 포함하는 DLL 만들기
  - 3개의 비트맵을 가지고 있는 DLL을 만든다.
  - 마우스를 클릭하면 화면의 임의의 위치에 3개의 비트맵이 번갈아 가면서 출력된다.

- 비트맵 리소스 DLL 만들기 위해서는
  - 비트맵 리소스만 포함하는 프로젝트를 만든다.
  - 빌드하기 전에 프로젝트->속성->링커->명령줄 메뉴에서 추가 옵션란에 "/NOENTRY" 옵션을 추가해준다.
  - 릴리즈 모드로 컴파일해서 *.dll 생성된 파일을 확인후 리소스 DLL 사용
  - 리소스 전용 DLL을 사용하는 응용 프로그램은 LoadLibrary를 호출하여 명시적으로 DLL을 호출해야 함

실습 11-1

- 사용예:

```
#include  “resource.h”

   HMODULE hRes = NULL;
   hRes = LoadLibrary(  “bitmap.dll");
   // hRes NULL이 아니면 성공

    hBitmap = LoadBitmap (hRes, MAKEINTRESOURCE(IDB_BITMAP1));
     // 비트맵 출력하는 코드 포함

     FreeLibrary (hRes);
```

실습 11-2