쉽게 배우는 MFC 윈도우 프로그래밍

Visual C++ 2019

2015·2017 버전 사용 가능

2장. MFC 기초 클래스

목차

실습 환경

- MFC 콘솔 응용 프로그램의 주요 장점
  - C/C++ 언어에 대한 지식만 있으면 곧바로 실습 가능
  - 상당수의 MFC 클래스 사용 가능
    - 유틸리티 클래스, 집합 클래스, 파일 입출력 클래스, 데이터베이스 클래스, 소켓 클래스, ...
  - 알고리즘 개발 시 유용
    - 프로그래밍이 간편한 콘솔 응응 프로그램을 많이 사용
    - printf( )와 같은 출력 함수를 사용할 수 있으므로 결과 확인 편리

MFC 콘솔 응용 프로그램 작성

■ 프로젝트 종류 선택

그림 2-3 프로젝트 종류 선택

MFC 콘솔 응용 프로그램 작성

■ 프로젝트 이름과 위치 지정

그림 2-4 프로젝트 이름과 위치 지정

MFC 콘솔 응용 프로그램 작성

■ 프로젝트 옵션 변경

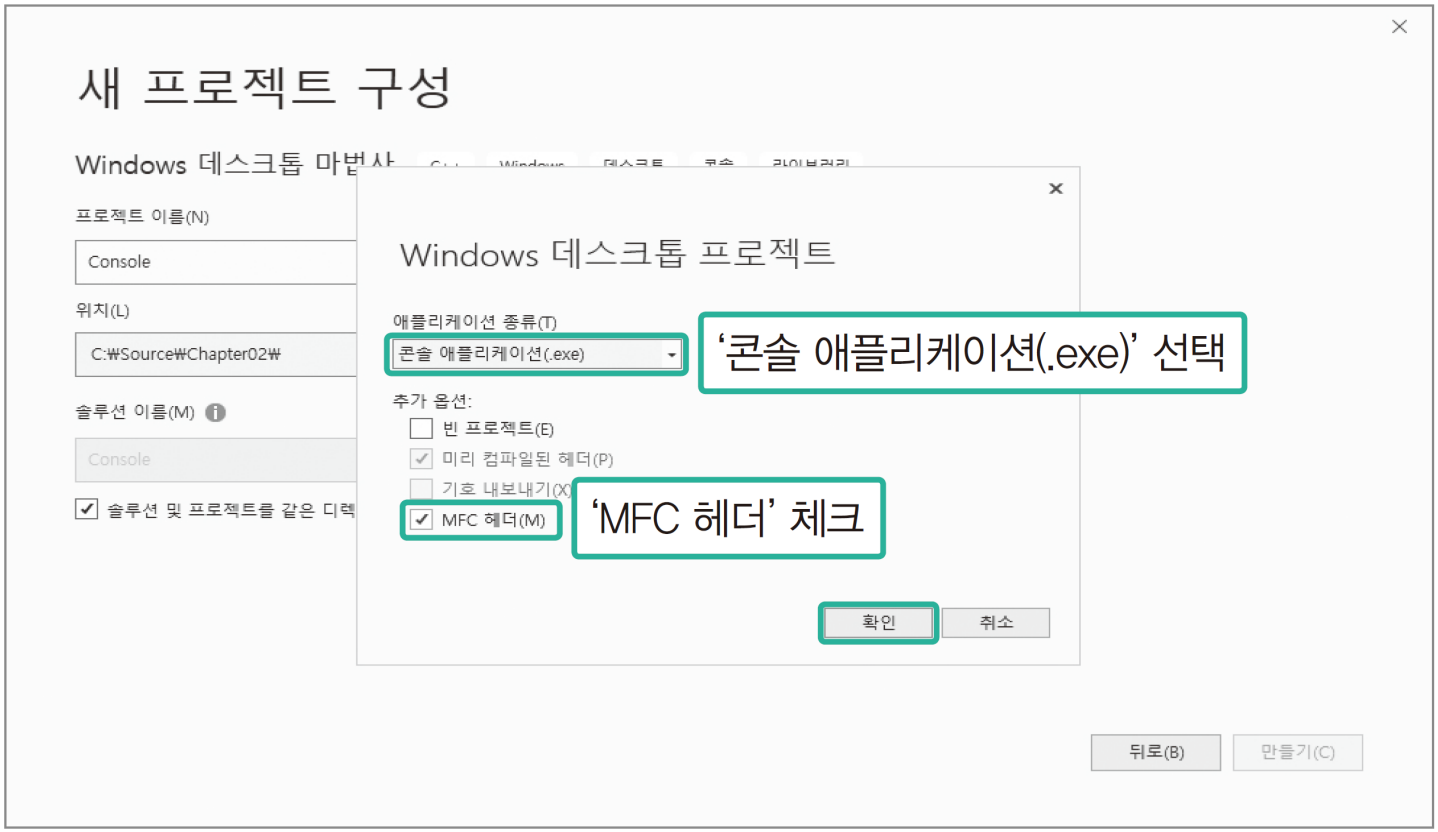

그림 2-5 프로젝트 옵션 변경

MFC 콘솔 응용 프로그램 작성

- 실행 결과

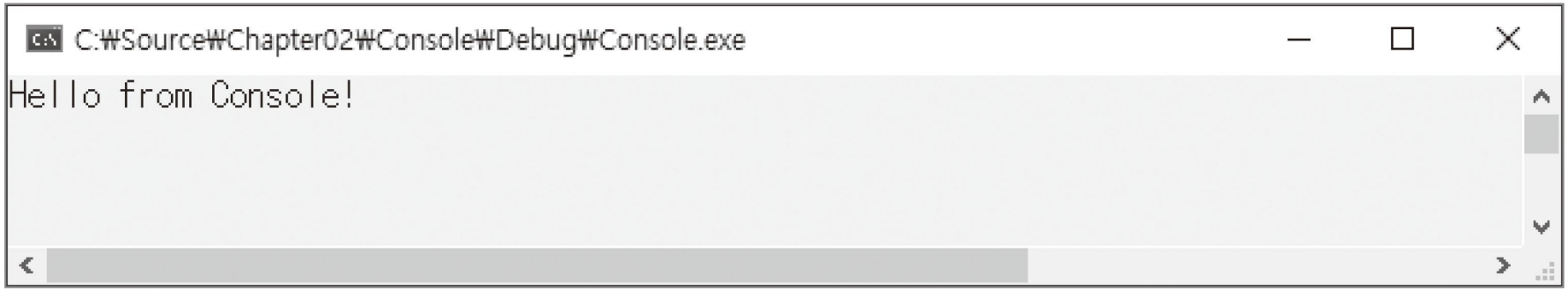

그림 2-2 실행 결과

MFC 콘솔 응용 프로그램 작성

- HelloMFC 예제 코드 (3/4)

```
int main()
{
    ...
    if (hModule != nullptr)
    {
        if (!AfxWinInit(hModule, nullptr, ::GetCommandLine(), 0))
        {
            wprintf(L"심각한 오류: MFC 초기화 실패\n");
            nRetCode = 1;
        }
        else
        {
            CString str;
            str.LoadString(IDS_APP_TITLE);
            _tprintf(_T("Hello from %s!\n"), (LPCTSTR)str);
            getchar();
        }
    }
    ...
}
```

MFC 콘솔 응용 프로그램 분석

예제 프로그램의 파일 구성

- framework.h, pch.h, pch.cpp
  - 미리 컴파일된 헤더 생성
- Console.rc, Resource.h
  - 윈도우 응용 프로그램에 포함해서 사용할 리소스의 목록은 *.RC 파일로 작성
  - 리소스 ID는 Resource.h 파일에 #define 으로 정의되어 있음
- Console.h, Console.cpp
  - 응용 프로그램 구현 코드가 있는 부분
- targetVer.h
  - 개발 대상이 되는 윈도우 운영체제의 버전을 지정하기 위한 헤더 파일

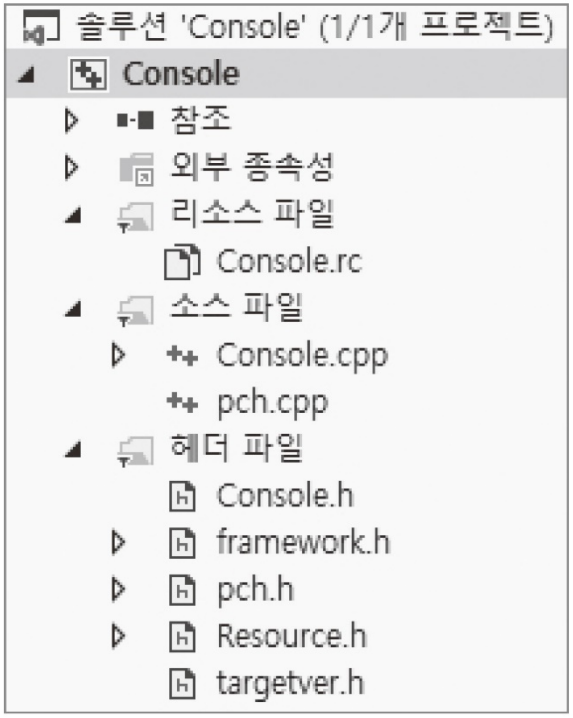

그림 2-6 예제 프로그램의 파일 구성

MFC 콘솔 응용 프로그램 분석

- Console 예제 코드 (1/2)

Console.cpp

```
#include "pch.h"
#include "framework.h"
#include "Console.h“

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

1 CWinApp theApp;                    //응용 프로그램 객체가 전역 변수로 정의되어 있음

using namespace std;

2 int main()                                  //전형적인 C 프로그램의 실행 시작점
{
   int nRetCode = 0;

   HMODULE hModule = ::GetModuleHandle(nullptr);

   if (hModule != nullptr)
   {
```

MFC 콘솔 응용 프로그램 분석

- Console 예제 코드 (2/2)

Console.cpp

```
3       if (!AfxWinInit(hModule, nullptr, ::GetCommandLine(), 0))
        {                                   //A fxWinInit( )는 MFC를 사용하기 위한 초기화 함수
          wprintf(L"심각한 오류: MFC 초기화 실패\n");
          nRetCode = 1;
        }
        else
        {
          CString str;
          str.LoadString(IDS_APP_TITLE);
          _tprintf(_T("Hello from %s!\n"), (LPCTSTR)str);
          getchar();
        }
    }
    else
    {
      wprintf(L"심각한 오류: GetModuleHandle 실패\n");
      nRetCode = 1;
    }
    return nRetCode;
}
```

MFC 콘솔 응용 프로그램 분석

- 문자열 리소스 추가

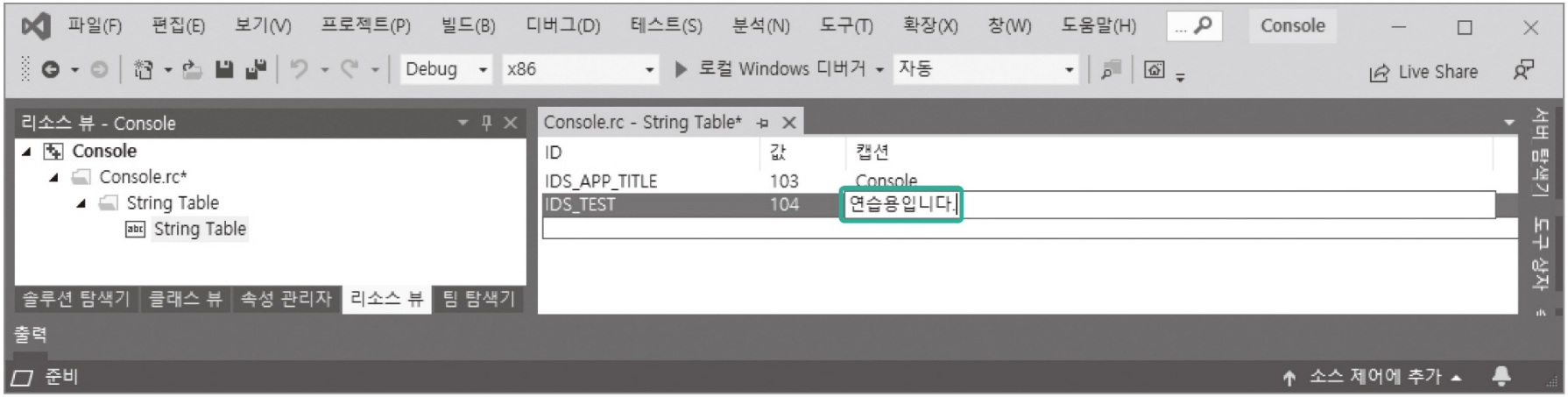

그림 2-7 문자열 리소스 추가

MFC 콘솔 응용 프로그램 분석

■ 실행 파일 생성 과정

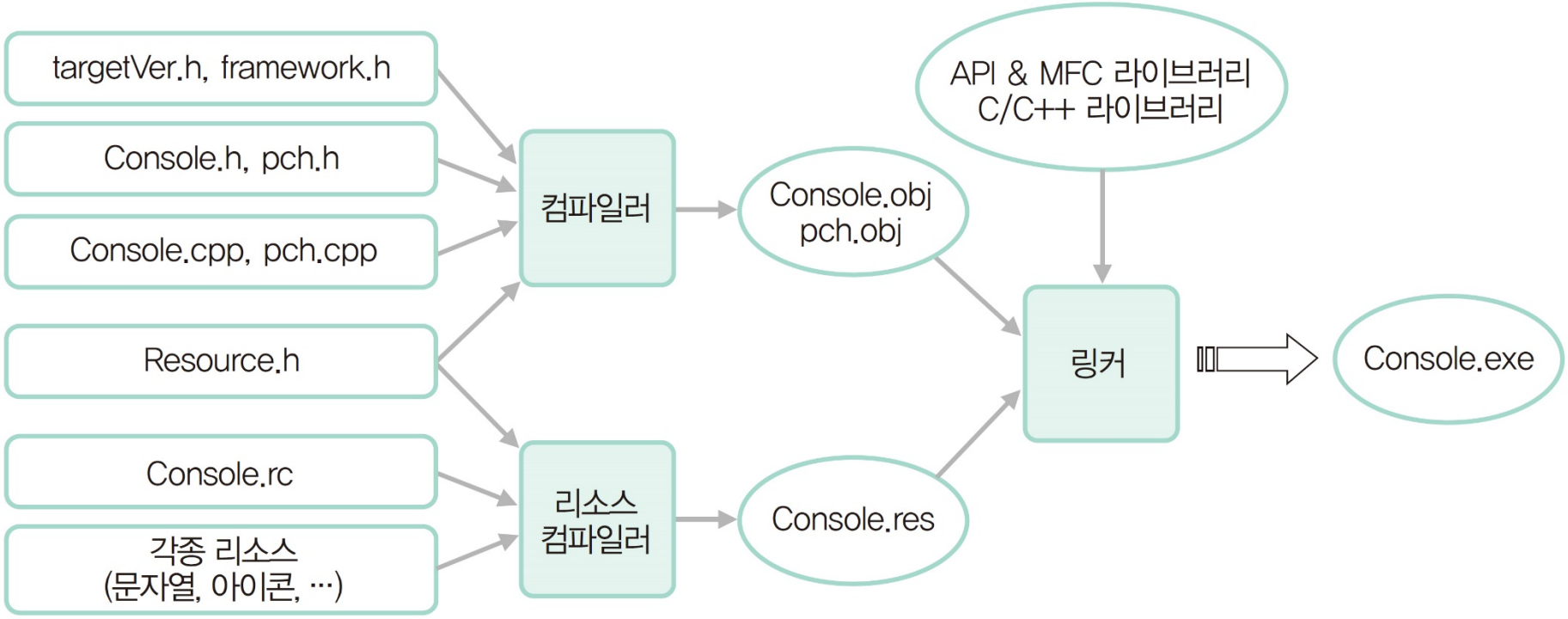

그림 2-8 실행 파일 생성 과정

API 데이터 타입

기본형

표 2-1 API 데이터 타입 : 기본형

데이터 타입	정의 또는 용도
BOOL 또는 BOOLEAN	TRUE 또는 FALSE
BYTE, WORD, DWORD, LONG	8비트, 16비트, 32비트, 32비트 (LONG을 제외하고는 모두 unsigned)
U*	unsigned* 예) UCHAR, UINT, ULONG, …
HANDLE	32비트(또는 64비트) 핸들
H*	*을 가리키는 핸들 예) HBITMAP(비트맵), HBRUSH(브러시), HCURSOR(커서), HDC(디바이스 컨텍스트), HFONT(폰트), HICON(아이콘), HINSTANCE(인스턴스), HMENU(메뉴), HPEN(펜), HWND(윈도우), …

API 데이터 타입

기본형

P* = LP*	*에 대한 포인터 예 PBOOL, PBYTE, PLONG, … 예 LPBOOL, LPBYTE, LPLONG, …
(L)PSTR, (L)PCSTR	ANSI 문자열(C는 const를 뜻함)
(L)PTSTR, (L)PCTSTR	ANSI 또는 유니코드 문자열(C는 const를 뜻함)
COLORREF	32비트 RGB(red, green, blue 각각 8비트) 색상값

API 데이터 타입

구조체

표 2-2 API 데이터 타입 : 구조체

데이터 타입	정의	용도
POINT	typedef struct tagPOINT { LONG x; LONG y; } POINT, *PPOINT;	점의 x, y 좌표
RECT	typedef struct tagRECT { LONG left; LONG top; LONG right; LONG bottom; } RECT, *PRECT;	직사각형의 왼쪽 상단과 오른쪽 하단 좌표
SIZE	typedef struct tagSIZE { LONG cx; LONG cy; } SIZE, *PSIZE;	폭과 높이

CString 클래스

- C/C++ 언어의 문자열 지원 방식
  - C 언어의 문자열 지원 방식
    - 문자열 타입을 따로 제공하지 않음
    - 문자 배열(0으로 끝나야 함)로 문자열 타입을 대신하고, 표준 라이브러리 형태로 문자열 조작 함수를 제공함
    - C 언어는 다른 언어에 비해 문자열을 다루기 불편함
  - C++ 언어의 문자열 지원 방식
    - C 언어와 마찬가지로 문자열 타입을 제공하지 않음
    - 표준 C++ 라이브러리(템플릿) 형태로 문자열 타입을 지원하므로 C 언어보다 편리하게 문자열을 다룰 수 있음

CString 클래스

- CString 클래스를 선호하는 이유
  - CString 클래스의 기능
    - CString 클래스는 일반 용도로도 사용하기 충분
    - C++ 표준 템플릿 라이브러리를 사용하면 코드 크기가 상대적으로 커질 수 있으므로 기능의 차이가 없다면 MFC가 제공하는 CString 클래스를 사용
  - CString 클래스와 MFC 함수의 연동
    - 문자열을 사용하는 많은 MFC 함수는 CString 객체(또는 객체의 주솟값)를 인자로 받으므로 CString 클래스로 처리한 문자열을 곧바로 MFC 함수 호출에 사용할 수 있음

CString 클래스

Cstring 클래스의 특징

- ANSI 또는 유니코드 문자열 지원
  - 프로젝트 속성에 따라 ANSI 또는 유니코드 문자열을 자동으로 지원

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
_tprintf(_T("유니코드로 저장된 한국어를 출력합니다.\n"));
```

- 가변 길이 문자열 지원
  - 프로그램 실행 중에 문자열 길이를 자유롭게 변경 가능
  - 문자열에 할당 가능한 최대 크기는 INT_MAX
- const TCHAR* 또는 LPCTSTR 대신 CString 객체를 직접 사용 가능

CString 클래스

- CString 객체 생성과 초기화

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
CString str1;
str1 = _T("안녕하세요."); // 문자열을 직접 대입한다.
CString str2(_T("오늘은")); // 문자열을 생성자 인자로 전달한다.
CString str3(str2); // CString 객체를 생성자 인자로 전달한다.
// CString 객체와 문자열을 붙인다.
CString str4 = str1 + _T(" ") + str2 + _T(" 즐거운 날입니다.");
_tprintf(_T("%s\n"), (LPCTSTR)str1);
_tprintf(_T("%s\n"), (LPCTSTR)str2);
_tprintf(_T("%s\n"), (LPCTSTR)str3);
_tprintf(_T("%s\n"), (LPCTSTR)str4);
// + = 연산자를 이용하여 기존 문자열에 새로운 문자열을 덧붙인다.
str4 += _T(" 하하하");
_tprintf(_T("%s\n"), (LPCTSTR)str4);
```

CString 클래스

- CString 객체 생성과 초기화

```
CString str;
str.Format(_T("x=%d, y=%d"), 100, 200);
MessageBox(NULL, str, _T("CString::Format() 연습"), MB_OK);
```

CString 클래스

- CString::Format() 함수

```
CString str;
str.Format(_T("x=%d, y=%d"), 100, 200);
MessageBox(NULL, str, _T("CString::Format() 연습"), MB_OK);
```

- CString::LoadString() 함수

```
CString str;
str.LoadString(IDS_APP_TITLE); // 문자열 리소스를 로드한다.
str.Insert(0, _T("Hello from ")); // 맨 앞에 문자열을 삽입한다.
str.Append(_T("!")); // 맨 끝에 문자열을 덧붙인다.
MessageBox(NULL, str, _T("CString::LoadString() 연습"), MB_OK);
```

CPoint, CRect, CSize 클래스

클래스 정의

표 2-3 클래스 정의

클래스 이름	정의
CPoint	class CPoint : public POINT { ... };
CRect	class CRect : public RECT { ... };
CSize	class CSize : public SIZE { ... };

업캐스팅

```
void SomeFunc(POINT pt); // 함수 원형 선언
POINT pt1 = { 100, 200 }; // POINT 구조체 변수 정의
CPoint pt2(300, 400); // CPoint 클래스 객체 정의
SomeFunc(pt1); // OK! (타입 일치)
SomeFunc(pt2); // OK! (업캐스팅)
```

CPoint 클래스

- CPoint 객체 생성과 다루기

```
CPoint::CPoint(int x, int y);
```

- 사용 예

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
CPoint pt1(10, 20); // x, y 좌표를 생성자 인자로 전달한다.
POINT pt = { 30, 40 };
CPoint pt2(pt); // POINT 타입 변수를 생성자 인자로 전달한다.
_tprintf(_T("%d, %d\n"), pt1.x, pt1.y);
_tprintf(_T("%d, %d\n"), pt2.x, pt2.y);
pt1.Offset(40, 30); // x, y 좌표에 각각 40, 30을 더한다.
pt2.Offset(20, 10); // x, y 좌표에 각각 20, 10을 더한다.
_tprintf(_T("%d, %d\n"), pt1.x, pt1.y);
_tprintf(_T("%d, %d\n"), pt2.x, pt2.y);
if (pt1 == pt2) // 두 객체의 내용이 같은지 확인한다.
    _tprintf(_T("두 점의 좌표가 같습니다.\n"));
else
    _tprintf(_T("두 점의 좌표가 다릅니다.\n"));
```

CRect 클래스

- CRect 객체 생성과 다루기
  1. 네 개의 멤버 변수값을 직접 생성자의 인자로 받는 것
  2. 직사각형의 폭과 높이를 곧바로 얻을 수 있는 Width( ) 와 Height( )
  3. 특정 좌표의 위치가 직사각형의 내부인지 외부인지 알려주는 PtInRect( ) 함수도 유용하다.

1. CRect::CRect(int l, int t, int r, int b);
2. int CRect::Width();

   int CRect::Height();
3. BOOL CRect::PtInRect(POINT point);

CRect 클래스

- 사용 예

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
CRect rect1(0, 0, 200, 100); // 직사각형의 좌표를 생성자의 인자로 전달한다.
CRect rect2;
rect2.SetRect(0, 0, 200, 100); // 직사각형의 좌표를 실행 중에 설정한다.
if (rect1 == rect2) // 두 객체의 내용이 같은지 확인한다.
    _tprintf(_T("두 직사각형의 좌표가 같습니다.\n"));
else
    _tprintf(_T("두 직사각형의 좌표가 다릅니다.\n"));
RECT rect = { 100, 100, 300, 200 };
CRect rect3(rect); // RECT 타입 변수를 생성자 인자로 전달한다.
_tprintf(_T("%d, %d\n"), rect3.Width(), rect3.Height());
CPoint pt(200, 150);
if (rect3.PtInRect(pt)) // 점이 직사각형 내부에 있는지 판단한다.
    _tprintf(_T("점이 직사각형 내부에 있습니다.\n"));
else
    _tprintf(_T("점이 직사각형 외부에 있습니다.\n"));
```

CSize 클래스

- CSize 객체 생성과 다루기

```
CSize::CSize(int x, int y);
```

- 사용 예

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
CSize size1(100, 200); // 폭과 높이를 생성자 인자로 전달한다.
SIZE size = { 100, 200 };
CSize size2(size); // SIZE 타입 변수를 생성자 인자로 전달한다.
_tprintf(_T("%d, %d\n"), size2.cx, size2.cy);
if (size1 == size2) // 두 객체의 내용이 같은지 확인한다.
    _tprintf(_T("크기가 같습니다.\n"));
else
    _tprintf(_T("크기가 다릅니다.\n"));
```

CTime, CTimeSpan 클래스

- CTime 객체 생성과 다루기
  - 절대적인 시간(예: 현재 시각) 처리
- 사용 예

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
// CTime::GetCurrentTime( ) 함수로 현재 시각을 구한다.
CTime tm;
tm = CTime::GetCurrentTime();
// 여러 형식으로 화면에 출력한다.
CString str = tm.Format(_T("%A, %B %d, %Y"));
_tprintf(_T("%s\n"), (LPCTSTR)str);
str.Format(_T("현재 시각은 %d시 %d분 %d초입니다."),
      tm.GetHour(), tm.GetMinute(), tm.GetSecond());
_tprintf(_T("%s\n"), (LPCTSTR)str);
```

CTime, CTimeSpan 클래스

- CTimeSpan 객체 생성과 다루기
  - 시간의 차이값을 처리
- 사용 예

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
CTime startTime = CTime::GetCurrentTime();
Sleep(2000); // 2000 밀리초 지연
CTime endTime = CTime::GetCurrentTime();
CTimeSpan elapsedTime = endTime - startTime;
CString str;
str.Format(_T("%d초 지남!"), elapsedTime.GetTotalSeconds());
_tprintf(_T("%s\n"), (LPCTSTR)str);
```

집합 클래스

MFC 집합 클래스

- 배열, 연결 리스트와 같은 자료 구조를 좀 더 편리하게 사용할 수 있도록 MFC에서 제공하는 클래스
- 집합 클래스는 크게 배열(Array), 리스트(List), 맵(Map)

배열 클래스

- 템플릿(Template) 클래스와 비템플릿(Nontemplate) 클래스 두 종류

집합 클래스

템플릿 클래스

- afxtempl.h 헤더 파일 필요

표 2-4 템플릿 배열 클래스

클래스 이름	데이터 타입	사용 예
CArray	프로그래머가 결정	CArray<CPoint, CPoint&> array;

집합 클래스

비템플릿 클래스

- afxcoll.h 헤더 파일 필요

표 2-5 비템플릿 배열 클래스

클래스 이름	데이터 타입	사용 예
CByteArray	BYTE	CByteArray array;
CWordArray	WORD	CWordArray array;
CDWordArray	DWORD	CDWordArray array;
CUIntArray	UINT	CUIntArray array;
CStringArray	CString	CStringArray array;
CPtrArray	void 포인터	CPtrArray array;
CObArray	CObject 포인터	CObArray array;

집합 클래스

배열 생성과 초기화

- MFC 배열 클래스를 이용한 배열 생성 순서
  - 배열 객체 생성
  - SetSize( ) 함수를 호출하여 크기 설정

```
CUIntArray array; // 객체를 생성한다.
array.SetSize(10); // 배열 크기를 설정한다.
for(int i=0; i<10; i++)
    array[i] = i*10; // 값을 대입한다.
for(int i=0; i<10; i++)
    _tprintf(_T("%d "), array[i]); // 값을 출력한다.
_tprintf(_T("\n"));
```

집합 클래스

- 배열 생성과 초기화
  - CStringArray 클래스를 이용하면 배열에 CString 객체 저장 가능

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요
CStringArray array; // 객체를 생성한다.
array.SetSize(5); // 배열 크기를 설정한다.
for (int i = 0; i < 5; i++) {
CString string;
string.Format(_T("%d년이 지났습니다."), (i + 1) * 10);
array[i] = string; // 값을 대입한다.
}
for (int i = 0; i < 5; i++)
_tprintf(_T("%s\n"), (LPCTSTR)array[i]); // 값을 출력한다.
```

집합 클래스

- 배열 원소 삽입과 삭제

```
CUIntArray array;
array.SetSize(5);
for (int i = 0; i < 5; i++)
	array[i] = i;
/* 배열 원소 삽입 */
array.InsertAt(3, 77); // 인덱스 3 위치에 원소를 삽입한다.
for (int i = 0; i < array.GetSize(); i++) // 변경된 배열의 크기만큼 반복한다.
	_tprintf(_T("%d "), array[i]);
_tprintf(_T("\n"));
/* 배열 원소 삭제 */
array.RemoveAt(4); // 인덱스 4 위치의 원소를 삭제한다.
for (int i = 0; i < array.GetSize(); i++)
	_tprintf(_T("%d "), array[i]);
_tprintf(_T("\n"));
```

배열 클래스

- 템플릿 배열 클래스 (1/3)

```
#include "pch.h"
#include "framework.h"
#include "ArrayTest2.h"
#include <afxtempl.h> // 템플릿 클래스 정의를 담고 있다.
...
// 3차원 좌표를 저장할 수 있는 클래스
// 모든 멤버가 public일 때는 class 대신 struct를 사용하면 좀 더 편리하다.
struct Point3D {
	int x, y, z;
	Point3D( ) {} // 템플릿 클래스에 사용할 때는 기본 생성자가 필요하다.
	Point3D(int x0, int y0, int z0) { x = x0; y = y0; z = z0; }
};
```

배열 클래스

- 템플릿 배열 클래스 (2/3)

```
int main()
{
    ...
        else
        {
            // Point3D 객체를 저장할 수 있는 배열 객체를 생성한다.
            CArray<Point3D, Point3D&> array;
            array.SetSize(5);
            for (int i = 0; i < 5; i++) {
                Point3D pt(i, i * 10, i * 100);
                array[i] = pt;
            }
```

배열 클래스

- 템플릿 배열 클래스 (2/3)

```
            for (int i = 0; i < 5; i++) {
                Point3D pt = array[i];
                _tprintf(_T("%d, %d, %d\n"), pt.x, pt.y, pt.z);
            }
        }
    }
    else
    {
        wprintf(L"심각한 오류: GetModuleHandle 실패\n");
        nRetCode = 1;
    }
    return nRetCode;
}
```

리스트 클래스

이중 연결 리스트

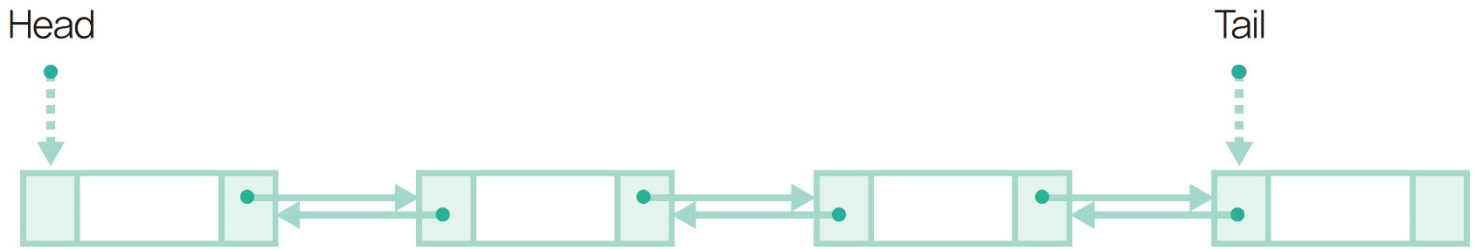

그림 2-9 이중 연결 리스트

템플릿 리스트 클래스

- afxtempl.h 헤더 파일 필요

표 2-6 템플릿 리스트 클래스

클래스 이름	데이터 타입	사용 예
CList	프로그래머가 결정	CList<CPoint, CPoint&> list;

리스트 클래스

■ 비템플릿 리스트 클래스

- afxcoll.h 헤더 파일 필요

표 2-7 비템플릿 리스트 클래스

클래스 이름	데이터 타입	사용 예
CObList	CObject 포인터	CObList list;
CPtrList	void 포인터	CPtrList list;
CStringList	CString 객체	CStringList list;

리스트 클래스

- 리스트 클래스를 이용한 리스트 생성/초기화 순서
  - 리스트 객체 생성
  - AddHead( ) 또는 AddTail( ) 함수를 호출하여 원소를 리스트의 앞쪽 또는 뒤쪽에 추가

```
// CString 객체는 물론이고 일반 문자열도 리스트에 추가할 수 있다.
TCHAR* szFruits[] = {
    _T("사과"), _T("딸기"), _T("포도"), _T("오렌지"), _T("자두")
};
CStringList list; // 리스트 객체를 생성한다.
for (int i = 0; i < 5; i++)
    list.AddTail(szFruits[i]); // 리스트 끝에 데이터를 추가한다.
```

리스트 클래스

리스트 순회

- 전방향 순회

```
POSITION pos = list.GetHeadPosition();
while (pos != NULL)
list.GetNext(pos); // 호출할 때마다 pos는 다음 데이터를 가리킨다.
```

- 후방향 순회

```
POSITION pos = list.GetTailPosition();
while (pos != NULL)
list.GetPrev(pos); // 호출할 때마다 pos는 이전 데이터를 가리킨다.
```

리스트 클래스

- 리스트 순회 예

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요

// 리스트 맨 앞에서부터 순회하면서 데이터를 출력한다.
POSITION pos = list.GetHeadPosition();
while (pos != NULL) {
	CString str = list.GetNext(pos);
	_tprintf(_T("%s "), (LPCTSTR)str);
}
_tprintf(_T("\n"));
// 리스트 맨 뒤에서부터 순회하면서 데이터를 출력한다.
pos = list.GetTailPosition();
while (pos != NULL) {
	CString str = list.GetPrev(pos);
	_tprintf(_T("%s "), (LPCTSTR)str);
}
_tprintf(_T("\n"));
```

리스트 클래스

- 리스트 항목 삽입과 삭제

```
pos = list.Find(_T("포도")); // 데이터의 위치를 얻는다.
list.InsertBefore(pos, _T("살구")); // 앞쪽에 데이터를 삽입한다.
list.InsertAfter(pos, _T("바나나")); // 뒤쪽에 데이터를 삽입한다.
list.RemoveAt(pos); // 데이터를 삭제한다.
// 리스트 맨 앞에서부터 순회하면서 데이터를 출력한다.
pos = list.GetHeadPosition();
while (pos != NULL) {
	CString str = list.GetNext(pos);
	_tprintf(_T("%s "), (LPCTSTR)str);
}
_tprintf(_T("\n"));
```

리스트 클래스

- 템플릿 리스트 클래스 (1/3)

```
#include "pch.h"
#include "framework.h"
#include "ListTest2.h"
#include <afxtempl.h> // 템플릿 클래스 정의를 담고 있다.
...
// 3차원 좌표를 저장할 수 있는 클래스
// 모든 멤버가 public일 때는 class 대신 struct를 사용하면 좀 더 편리하다.
struct Point3D {
	int x, y, z;
	Point3D( ) {} // 템플릿 클래스에 사용할 때는 기본 생성자가 필요하다.
	Point3D(int x0, int y0, int z0) { x = x0; y = y0; z = z0; }
};
```

리스트 클래스

- 템플릿 리스트 클래스 (2/3)

```
int main()
{
    ...
        else
        {
            // Point3D 객체를 저장할 수 있는 리스트 객체를 생성한다.
            CList<Point3D, Point3D&> list;

            // 리스트 끝에 데이터를 추가한다.
            for (int i = 0; i < 5; i++)
                list.AddTail(Point3D(i, i * 10, i * 100));

            // 리스트 맨 앞에서부터 순회하면서 데이터를 출력한다.
            POSITION pos = list.GetHeadPosition();
```

리스트 클래스

- 템플릿 리스트 클래스 (3/3)

```
            while (pos != NULL) {
                Point3D pt = list.GetNext(pos);
                _tprintf(_T("%d, %d, %d\n"), pt.x, pt.y, pt.z);
            }
        }
    }
    else
    {
        wprintf(L"심각한 오류: GetModuleHandle 실패\n");
        nRetCode = 1;
    }

    return nRetCode;
}
```

맵 클래스

- 맵의 동작 원리

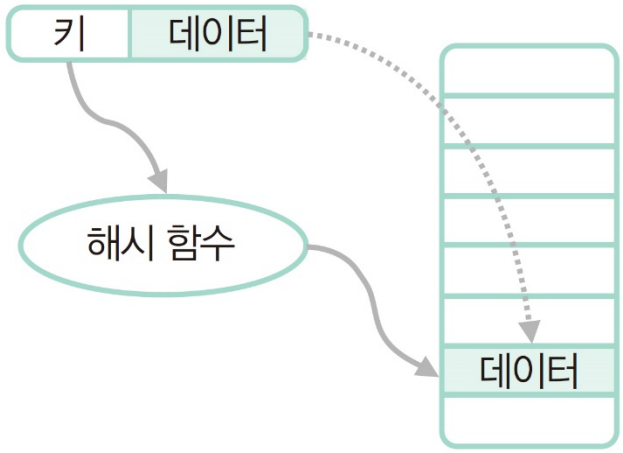

그림 2-10 맵의 동작 원리

- MFC의 맵 구현

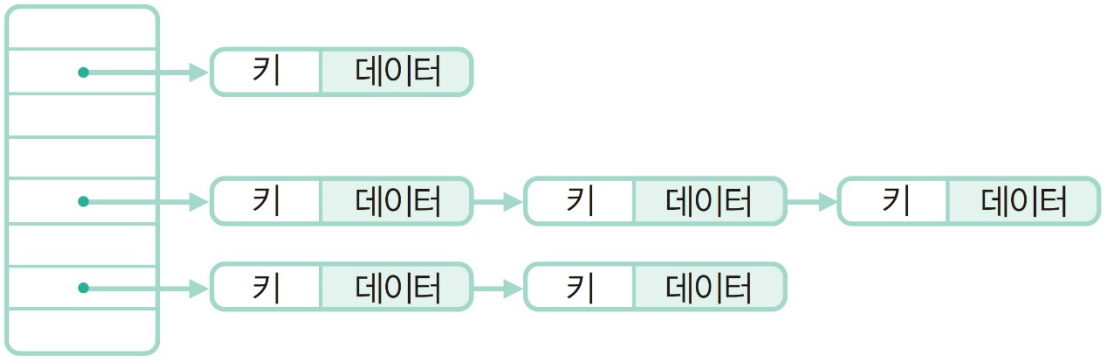

그림 2-11 MFC의 맵 구현

맵 클래스

■ 템플릿 맵 클래스

- afxtempl.h 헤더 파일 필요

표 2-8 템플릿 맵 클래스

클래스 이름	데이터 타입	사용 예
CMap	프로그래머가 결정	CMap<CString, CString&, CPoint, CPoint&> map;

맵 클래스

■ 비템플릿 맵 클래스

- afxcoll.h 헤더 파일 필요

표 2-9 비템플릿 맵 클래스

클래스 이름	데이터 타입(키 → 데이터)	사용 예
CMapWordToOb	WORD → CObject 포인터	CMapWordToOb map;
CMapWordToPtr	WORD → void 포인터	CMapWordToPtr map;
CMapPtrToWord	void 포인터 → WORD	CMapPtrToWord map;
CMapPtrToPtr	void 포인터 → void 포인터	CMapPtrToPtr map;
CMapStringToOb	문자열 → CObject 포인터	CMapStringToOb map;
CMapStringToPtr	문자열 → void 포인터	CMapStringToPtr map;
CMapStringToString	문자열 → 문자열	CMapStringToString map;

맵 클래스

맵 생성과 초기화 및 검색

- 맵 객체 생성
- [ ] 연산자를 이용한 맵 초기화(맵 객체[키] = 데이터)
- 맵 객체.Lookup(키, 검색된 데이터를 담을 변수) 형식으로 함수를 호출하여 특정 키값을 가진 데이터 검색해 데이터를 얻음

```
_tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요

// 맵 객체를 생성하고 초기화한다.
CMapStringToString map;
map[_T("사과")] = _T("Apple");
map[_T("딸기")] = _T("Strawberry");
map[_T("포도")] = _T("Grape");
map[_T("우유")] = _T("Milk");
// 특정 키 값을 가진 데이터를 검색한다.
CString str;
if (map.Lookup(_T("딸기"), str))
    _tprintf(_T("딸기 -> %s\n"), (LPCTSTR)str);
```

맵 클래스

맵 순회

```
POSITION pos = map.GetStartPosition();
while (pos != NULL)
    map.GetNextAssoc(pos, 키값을 저장할 변수, 데이터값을 저장할 변수);
```

맵 순회 예제

```
_tprintf(_T("\n"));
// 맵을 순회하면서 모든 키와 데이터값을 출력한다.
POSITION pos = map.GetStartPosition();
while (pos != NULL) {
    CString strKey, strValue;
    map.GetNextAssoc(pos, strKey, strValue);
    _tprintf(_T("%s -> %s\n"), (LPCTSTR)strKey,
    (LPCTSTR)strValue);
}
```

맵 클래스

- 맵 데이터 삽입과 삭제

```
_tprintf(_T("\n"));
map.RemoveKey(_T("우유")); // 키값 "우유"에 해당하는 데이터를 삭제한다.
map[_T("수박")] = _T("Watermelon"); // map.SetAt(_T("수박"), _T("Watermelon"));
// 맵을 순회하면서 모든 키와 데이터값을 출력한다.
pos = map.GetStartPosition();
while (pos != NULL) {
    CString strKey, strValue;
    map.GetNextAssoc(pos, strKey, strValue);
    _tprintf(_T("%s -> %s\n"), (LPCTSTR)strKey,
    (LPCTSTR)strValue);
}
```

맵 클래스

- 맵 최적화

```
CMapStringToString map; // 맵 객체를 생성한다.
map.InitHashTable(12007); // 해시 테이블 크기를 12007로 바꾼다.
```

맵 클래스

- 템플릿 맵 클래스 (1/3)

```
#include "pch.h"
#include "framework.h"
#include "MapTest2.h"
#include <afxtempl.h> // 템플릿 클래스 정의를 담고 있다.
...
// CString 타입에 해당하는 해시 함수가 없으므로 정의한다.
template <> UINT AFXAPI HashKey(CString& str)
{
    LPCTSTR key = (LPCTSTR)str;
    return HashKey(key); // LPCTSTR 타입의 해시 함수를 재호출한다.
}
```

맵 클래스

- 템플릿 맵 클래스 (2/3)

```
int main()
{
    ...
        else
        {
            _tsetlocale(LC_ALL, _T("")); // 유니코드 한국어 출력에 필요

            // 맵(CString -> UINT) 객체를 생성하고 초기화한다.
            CMap<CString, CString&, UINT, UINT&> map;
            map[CString(_T("사과"))] = 10;
            map[CString(_T("딸기"))] = 25;
```

맵 클래스

- 템플릿 맵 클래스 (3/3)

```
            map[CString(_T("포도"))] = 40;
            map[CString(_T("수박"))] = 15;
            // 특정 키값을 가진 데이터를 검색한다.
            UINT nCount;
            if (map.Lookup(CString(_T("수박")), nCount))
                _tprintf(_T("수박 %d상자가 남아 있습니다.\n"), nCount);
        }
    }
    else
    {
        wprintf(L"심각한 오류: GetModuleHandle 실패\n");
        nRetCode = 1;
    }
    return nRetCode;

}
```

맵 클래스

- 비주얼 C++ 2019의 HashKey( ) 정의

```
template<class ARG_KEY>
AFX_INLINE UINT AFXAPI HashKey(ARG_KEY key)
{
	ldiv_t HashVal = ldiv((long)(ARG_KEY)key, 127773);
	HashVal.rem = 16807 * HashVal.rem - 2836 * HashVal.quot;
	if (HashVal.rem < 0)
		HashVal.rem += 2147483647;
	return ((UINT)HashVal.rem);
}
```

```
UINT AFXAPI HashKey(CString& str);
```