유틸리티 클래스와 집합 클래스

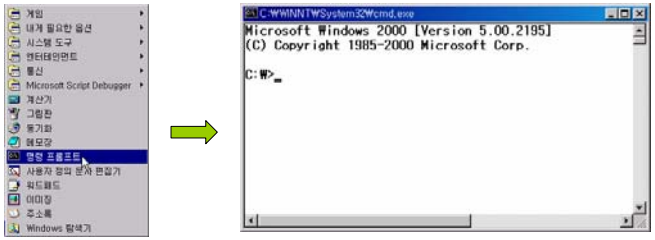
HCI Programming 2 (321190)
2007년 가을학기
9/20/2007
박경신

Overview

- 유틸리티 클래스를 이용하여 객체 생성법과 사용법
- MFC에서 C++의 업캐스팅이 적용되는 원리 이해
- 배열, 리스트, 맵 클래스 동작 원리와 사용법

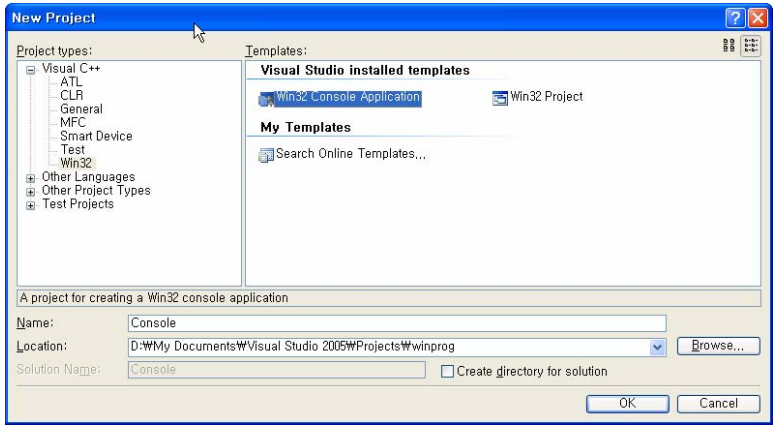
콘솔 응용 프로그램

- 특징
  - 메시지 구동 방식을 사용하지 않으므로 C/C++ 언어에 대한 지식만 있으면 곧바로 실습이 가능하다.
  - 상당수의 MFC 클래스를 사용할 수 있다.
    - 유틸리티 클래스, 집합 클래스, 파일 입출력 클래스, ...
  - 알고리즘을 개발할 때 유용하다.



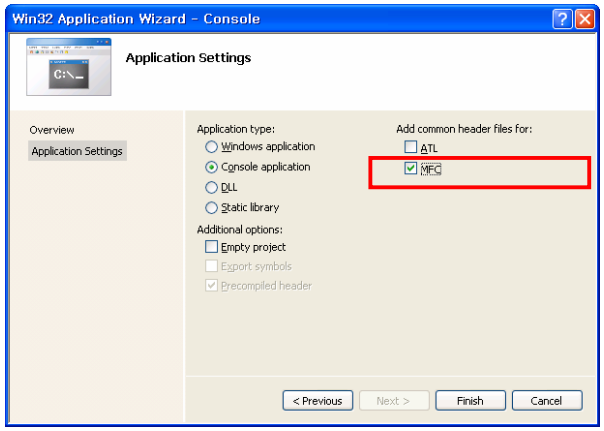
MFC 콘솔 응용 프로그램 작성

- 프로젝트 생성



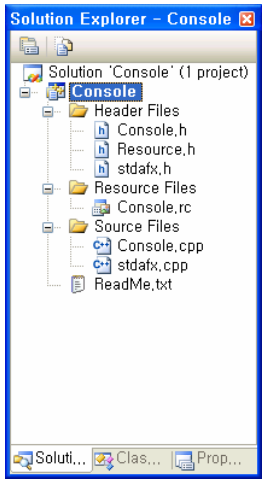
MFC 콘솔 응용 프로그램 작성

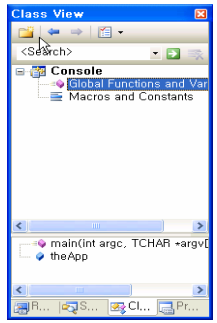
- 1단계 옵션 설정

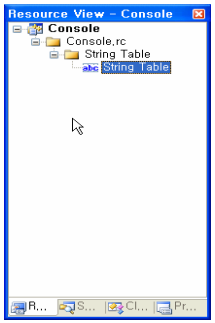


MFC 콘솔 응용 프로그램 분석

- 파일 구성
  - stdafx.h, stdafx.cpp
    - 미리 컴파일된 헤더
  - Console.rc, Resource.h
    - 리소스 정보
  - Console.h, Console.cpp
    - 구현 코드







MFC 콘솔 응용 프로그램 분석

- stdax.f.h
  - 미리 컴파일 된 헤더 생성
  - MFC에서 사용하는 헤더화일은 크기가 커서 컴파일 시간이 오래 걸림
  - 사용할 헤더 파일을 stdafx.h에 선언하면 컴파일러가 헤더를 미리 컴파일하여 *.pch파일로 저장함. 다음 컴파일때 시간이 단축할 수 있음.
- Console.rc
  - 프로그램에서 사용할 리소스는 *.rc파일로 작성됨
  - Resource view 에서 볼 수 있음
- Resource.h
  - 프로그램에서 해당 리소스를 MACRO상수 값으로 참조 가능

MFC 콘솔 응용 프로그램 분석

- Console.cpp 코드
  - CString객체 생성하고, CString 멤버함수인 LoadString()을 이용하여 리소스에서 문자열을 load한 후 cout을 이용하여 화면에 출력

```
CWinApp theApp; // 유일한 전역 응용 프로그램 객체
                // CWinApp 클래스를 그대로 이용
using namespace std; // 표준 C++ 라이브러리를 사용

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
   int nRetCode = 0;

   if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine( ), 0))
   {
      cerr << _T("Fatal Error: MFC initialization failed") << endl;
      nRetCode = 1;
   }
```

MFC를 사용하기 위한 초기화 함수

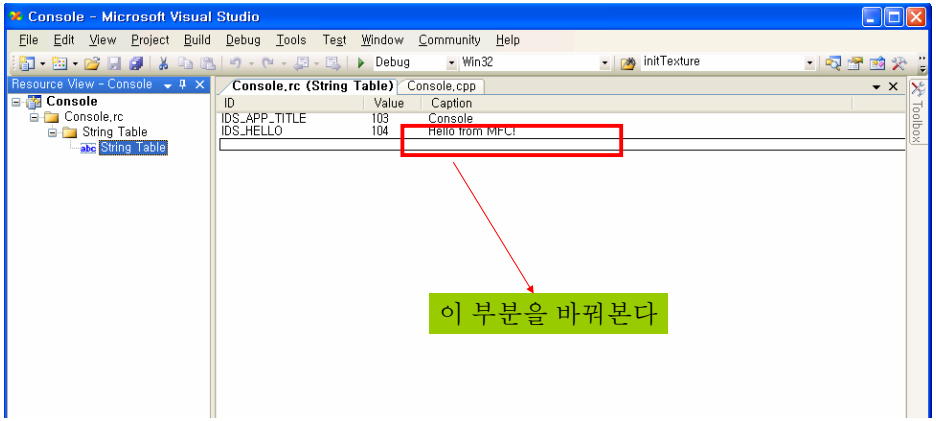
MFC 콘솔 응용 프로그램 분석

- Console.cpp

```
    else
    {
        CString strHello;      // CString 객체를 생성
        strHello.LoadString(IDS_HELLO);   // 리소스에서 문자열을 load
        cout << (LPCTSTR)strHello << endl;          // 화면 출력
    }

    return nRetCode;
}
```

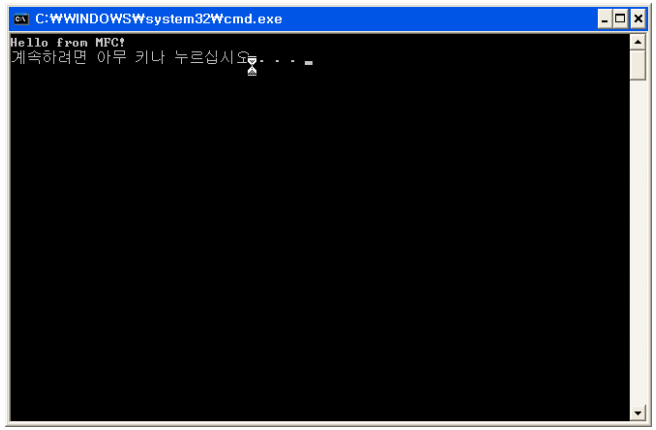

MFC 콘솔 응용 프로그램 분석

- 문자열 리소스



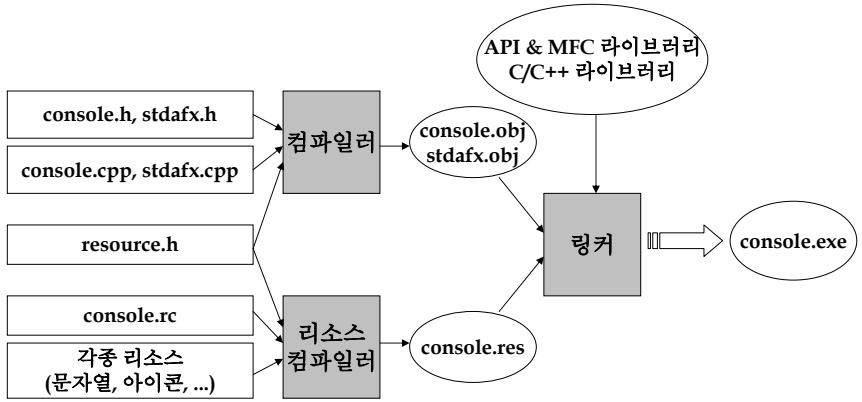
MFC 콘솔 응용 프로그램 분석

- 실행 (CTRL+F5)



MFC 콘솔 응용 프로그램 분석

- 실행 파일 생성 과정



Data Type & Data Type Class

- 변수 명명법
- API 데이터 타입
- 데이터 타입 클래스
  - CString
  - CPoint, CRect, CZize
  - CTime, CTimeSpan
- 집합 클래스
  - CArray
  - CList
  - CMap

변수 명명법

- 윈도우 프로그램에서는 변수명을 헝가리언 표기법을 사용
- 변수이름 앞에 데이터 형식을 의미하는 접두어를 사용하여 명명
- 변수의 형식에 대한 불일치 오류방지
- 프로그램 해독, 유지, 보수에 용이

접두어	데이터 타입	사용 예
a	Array	char achName[10]
by	BYTE (unsigned char)	BYTE byCh
b	BOOL(int)	BOOL bFind
ch	Char	char chValue
cb	Count of Bytes	BYTE cbLoop
dw	DWORD(unsigned long)	DWORD dwSum
fn	Function	void fnGetData()
h	Handle	HDC hdc , HWND hwnd
i	Index	int iName
l	LONG	LONG lParam
lp	Long (or far) pointer	LPSTR lpBuffer
n	Short or int	int n
np	Near(or short) pointer	char * npStr
pt	POINT(x, y pointer)	POINT ptStr
r	RECT(Rectangelstructure)	RECT rScreen
s	String	char sAdderss[40]
sz	Null-teminated string	char szMsg[]
m	data member in Class	LONG m_ID
w	WORD(unsigned int)	WORD wNum

API Data Type

- API 데이터 타입 – 기본형 (windef.h)

데이터 타입	정의
BOOL or BOOLEAN	TRUE or FALSE
BYTE, WORD, DWORD, LONG	8비트, 16비트, 32비트, 32비트(모두 unsigned)
U*	unsigned * 예) UCHAR, UINT, ULONG, ...
HANDLE	32비트 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 Data Type

- API 데이터 타입 – 기본형 (windef.h)

데이터 타입	정의
P* = LP*	* 에 대한 포인터 예1) PBOOL, PBYTE, PLONG, ... 예2) LPBOOL, LPBYTE, LPLONG, ...
(L)PSTR, (L)PCSTR	ANSI 문자열
(L)PTSTR, (L)PCTSTR	ANSI 또는 유니코드 문자열
COLORREF	32비트 RGB (red, green, blue 각각 8비트) 색상값

Unicode

- ASCII code
  - 8-bit로 표현할 수 있는 256자
  - 영어나 라틴어권에서는 상관없음
  - 한국, 일본, 중국, 아랍 등의 다양한 문자들을 표현하는 데는 한계
- Unicode
  - 세계 각국의 언어를 통일된 방법으로 표현할 수 있게 제안된 국제적인 코드 규약
  - 비트 문자코드인 ASCII 코드를 16-bit로 확장
    - 전 세계의 모든 문자를 표현하는 표준 코드
    - 11,172자의 한글을 연속된 공간에 가나다라 순서로 '가'에서 'ㅎ'까지 코드화하는 방식이 유니코드기술위원회 (UTC)에서 채택한 유니코드 2.0 규격임.
    - Windows NT, C언어, Java's native encoding이 유니코드 지원
  - 장점- 하나의 캐릭터로 표현
  - 단점 - 메모리 공간을 두 배 차지

API Data Type

- API 데이터 타입 - 구조체

데이터 타입	정의	용도
POINT	typedef struct tagPOINT { LONG x; LONG y; } POINT, *PPOINT;	점의 x, y 좌표(주로 마우스 커서의 위치를 나타낼 때 사용한다.)
RECT	typedef struct tagRECT { LONG left; LONG top; LONG right; LONG bottom; } RECT, *PRECT;	사각형 왼쪽 상단과 오른쪽 하단 좌표
SIZE	typedef struct tagSIZE { LONG cx; LONG cy; } SIZE, *PSIZE;	사각형 폭과 높이

유틸리티 클래스

- 데이터 타입 Class
  - CString
  - CPoint
  - CRect
  - CSize
  - CTime
  - CTimeSpan

CString 클래스

- 기능
  - 문자열 처리
  - 대부분의 MFC 지원 함수처럼 dll로 제공
    - 코드의 크기가 줄어 듬
  - 다양한 초기화 방법, 멤버함수, 연산자 등을 지원
    - 손쉽게 문자를 처리

CString 클래스

- 특성
  - 가변 길이 문자열을 지원
    - 프로그램 실행 도중 문자열 길이를 자유롭게 바꿀 수 있다. 최대 길이는 (INT_MAX - 1)이다.
  - const char*나 LPCTSTR 대신 CString 객체를 직접 사용 가능
    - 이때는 CString 객체에 대해 (LPCTSTR) 연산자를 명시적으로 적용하는 것이 좋다.

```
CString str = "안녕하세요.";
cout << (LPCTSTR)str << endl; // 실행 결과?
cout << str << endl; // 실행 결과?
```

(LPCTSTR)을 제거하면 Unicode로 casting이 되지 않음

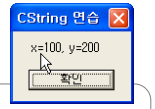
CString 클래스

- 객체 생성 및 초기화
  - Console.cpp에서 다음을 추가해 본다.

```
CString str1;
str1 = "안녕하세요.";
CString str2("오늘은");
CString str3(str2);
CString str4 = str1 + " " + str2 + " 즐거운 날입니다.";
cout << (LPCTSTR)str1 << endl;
cout << (LPCTSTR)str2 << endl;
cout << (LPCTSTR)str3 << endl;
cout << (LPCTSTR)str4 << endl;
str4 += " 하하하";
cout << (LPCTSTR)str4 << endl;
```


CString 클래스

- CString::Format( )
  - sprintf와 같이 데이터를 문자열로 변환하여 출력




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

- CString::LoadString( )
  - 문자열 테이블로부터 문자열 리소스를 로드


```
CString str;
str.LoadString(IDS_HELLO);
MessageBox(NULL, (LPCTSTR)str, "CString 연습", MB_OK);
```

CString 클래스

- 문자열 변환
  - int CString::Replace(const char* s1, const char* s2)
  - 객체의 문자열 중 모든 s1을 s2로 변경 후 변경된 횟수 반환
- 문자열 검색
  - int CString::Find(const char* s)
  - 객체의 문자열에서 s를 찾아 인덱스 위치를 리턴
- 문자열 삭제
  - int CString::Delete(int index, int count)
  - index 위치에서 count 개의 문자를 제거후 제거된 문자 수 반환
- 문자열 삽입
  - int CString::Insert(int index, const char* s)
  - index 위치에 s를 삽입
- 문자열 길이
  - int GetLength( )
- 문자열 초기화
  - void Empty( );
- 문자열 추출
  - CString Mid(int *nFirst*)
  - CString Mid(int *nFirst*, int *nCount*)
  - CString Left(int *nCount*)
  - CString Right(int *nCount*)

CPoint, CRect, CSize 클래스

- 클래스 정의

클래스 이름	정의
CPoint	class CPoint : public POINT
CRect	class CRect : public RECT
CSize	class CSize : public SIZE

- 업캐스팅
  - 하위클래스가 상위클래스로 형 변환

```
void SomeFunc(RECT* rect) { ... }
RECT r1; CRect r2;
SomeFunc(&r1); // OK!
SomeFunc(&r2); // OK! (Upcasting)
```


CPoint 클래스

- 생성자

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

- 예제

```
CPoint pt1(10, 20);
POINT pt = {30, 40};
CPoint pt2(pt);
pt1.Offset(40, 30);        // 40, 30을 더함
pt2.Offset(20, 10);        // 20, 10을 더함
if(pt1 == pt2)
   cout << "두 점의 좌표가 같습니다." << endl;
else
   cout << "두 점의 좌표가 다릅니다." << endl;
```


CRect 클래스

- 생성자

```
CRect::CRect (int l, int t, int r, int b);
CRect::CRect (const RECT& srcRect);
CRect::CRect (LPCRECT lpSrcRect);
CRect::CRect (POINT point, SIZE size);
CRect::CRect (POINT topLeft, POINT bottomRight);
```

- 사각형의 폭과 높이

```
int CRect::Width ();
int CRect::Height ();
```

- 좌표가 사각형의 내부인지 외부인지 포함 여부 판단

```
BOOL CRect::PtInRect (POINT point);
```

CRect 클래스

- 예제

```
CRect rect1;
rect1.SetRect(0, 0, 200, 100);
CRect rect2(0, 0, 200, 100);
if(rect1 == rect2)
   cout << "두 사각형의 좌표가 같습니다." << endl;
else
   cout << "두 사각형의 좌표가 다릅니다." << endl;

RECT rect = {100, 100, 300, 200};
CRect rect3(rect);
cout << rect3.Width() << " " << rect3.Height() << endl;

CPoint pt(200, 150);
if(rect3.PtInRect(pt))
   cout << "점이 사각형 내부에 있습니다." << endl;
else
   cout << "점이 사각형 외부에 있습니다." << endl;
```


CSize 클래스

- 생성자

```
CSize::CSize (int x, int y);
CSize::CSize (SIZE initSize);
CSize::CSize (POINT initPoint);
CSize::CSize (DWORD dwSize);
```

- 예제

```
CSize size1(100, 200);  // 폭과 높이 지정
SIZE size = {100, 200};
CSize size2(size);
cout << size2.cx << " " << size2.cy << endl;
if(size1 == size2) // 두 객체 내용이 같은지 확인한다.
   cout << "크기가 같습니다." << endl;
else
   cout << "크기가 다릅니다." << endl;
```


CTime, CTimeSpan 클래스

- CTime 클래스 (afx.h)
  - 절대적인 시간(예를 들면, 현재 시각)을 처리
- CTimeSpan 클래스 (afx.h)
  - 시간의 차이 값을 처리
- 두 클래스 모두 내부적으로 시간 값을 64비트로 저장
- 생성자

```
CTime::CTime (const CTime& timeSrc );
    CTime::CTime (time_t time );
    CTime::CTime (int nYear, int nMonth, int nDay, int nHour, int nMin,
                  int nSec, int nDST = -1 );
    CTime::CTime (WORD wDosDate, WORD wDosTime, int nDST = -1 );
```


CTime 클래스

- 예제

```
// 현재 시각 구하기
CTime theTime;
theTime = CTime::GetCurrentTime();

// 화면에 출력하기
CString str = theTime.Format("%A, %B %d, %Y");
cout << (LPCTSTR)str << endl;
str.Format("현재 시각은 %d시 %d분입니다.",
   theTime.GetHour(),
   theTime.GetMinute());
cout << (LPCTSTR)str << endl;
```

요일, 월 날짜, 연도

CTimeSpan 클래스

- 예제

```
// 시간 차 구하기
CTime startTime = CTime::GetCurrentTime();
Sleep(2000);   // 2 sec
CTime endTime = CTime::GetCurrentTime();
CTimeSpan elapsedTime = endTime - startTime;
CString str;
str.Format("%d초가 지났습니다.", elapsedTime.GetTotalSeconds());
cout << (LPCTSTR)str << endl;
```


집합 클래스

- 집합 클래스 (Collection Class)
  - Array, linked list, map과 같은 자료 구조를 좀 더 편리하게 사용할 수 있도록 MFC에서 제공하는 클래스
  - Array
  - List
  - Map

배열

- 배열 (Array)의 단점
  - 원소를 삽입하거나 삭제시 인접 원소를 이동해야하므로 속도 저하
  - 배열 인덱스를 잘못 참조해도 오류를 발생시키지 않음
  - 배열의 크기가 고정적
- MFC는 이런 문제 해결을 위하여 멤버함수를 추가한 다양한 종류의 배열 클래스를 제공
  - Template 클래스
  - Nontemplate 클래스

배열 클래스

- MFC 배열 클래스의 특징
  - 배열 인덱스를 잘못 참조하는 경우 오류를 발생시킨다.
  - 배열 크기가 가변적이다.
- 템플릿 클래스
  - afxtempl.h 헤더 파일
  - 원하는 종류의 데이터 타입을 프로그래머가 결정

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

배열 클래스

- 비 템플릿 클래스
  - afxcoll.h 헤더 파일
  - 각 클래스마다 데이터 타입이 있다

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;

배열 클래스

- 생성과 초기화
  - 배열 객체를 생성
  - SetSize를 호출하여 크기를 설정
  - 초기화/접근 시 [] 연산자를 이용

```
CStringArray array;
array.SetSize(5);
for(int i=0; i<5; i++){
   CString string;
   string.Format("%d년이 지났습니다.", i*10);
   array[i] = string;
}
for(i=0; i<5; i++)
   cout << (LPCTSTR)array[i] << endl;
```


배열 클래스

- 원소 삽입과 삭제

```
CUIntArray array;
array.SetSize(5);
for(int i=0; i<5; i++)
   array[i] = i;
// 배열 원소 삽입 InsertAt(index, 값)
array.InsertAt(3, 77);\
for(i=0; i<array.GetSize(); i++)    // 배열의 현재 크기 GetSize()
   cout << array[i] << endl;
cout << endl;
// 배열 원소 삭제 Remove(index)
array.RemoveAt(3);
for(i=0; i<array.GetSize(); i++)
   cout << array[i] << endl;
```


배열 클래스

- 템플릿 배열 클래스
  - 사용자가 정의한 데이터 타입인 경우 (즉, MFC에서 제공하지 않는 데이터 타입의 경우)에 사용

```
#include "stdafx.h"
#include "Console.h"
#include <afxtempl.h>   // 템플렛 클래스 사용을 위해 필요

CWinApp theApp;

using namespace std;

struct Point3D {
   int x, y, z;
   Point3D() {}   // 템플렛의 경우 반드시 기본 생성자가 필요
   Point3D(int x0, int y0, int z0) { x = x0; y = y0; z = z0; }
};
```

배열 클래스

```
int _tmain(int argc, TCHAR* argv[ ], TCHAR* envp[ ])
{
    int nRetCode = 0;

    if (!AfxWinInit(...))
    {
        // 생략 ...
    }
    else
    {
        // Point3D 객체를 저장할 수 있는 배열 객체 생성
        CArray<Point3D, Point3D&> array;
        array.SetSize(5);               // 크기 생성
        for(int i=0; i<5; i++) {
            Point3D pt(i, i*10, i*100); // x, y, z축의 좌표값을 동시에 배열에 저장
            array[i] = pt;              // 배열에 pt 대입
        }
```


배열 클래스

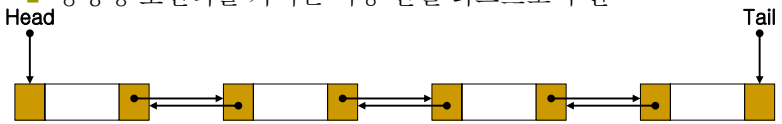
```
        for(i=0; i<5; i++) {
            Point3D pt = array[i];
            cout << pt.x << ", " << pt.y << ", " << pt.z << endl;
        }
    }

    return nRetCode;
}
```


리스트

- Linked List
  - 동일한 데이터 타입의 데이터를 포인터를 이용하여 연결시킨 자료 구조
  - 장점
    - 원소의 삽입과 삭제가 포인터 조작만으로 가능하기 때문에 빠르다
  - 단점
    - 특정 위치에 있는 원소를 참조시, 리스트의 처음 또는 끝에서 시작해서 포인터를 따라가야 하므로 속도가 느려진다

리스트 클래스

- MFC 리스트 클래스
  - 양방향 포인터를 가지는 이중 연결 리스트로 구현



- 템플릿 클래스
  - afxtempl.h 헤더 파일

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

리스트 클래스

- 비 템플릿 클래스
  - afxcoll.h 헤더 파일

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

리스트 클래스

- 생성과 초기화 - 비템플릿

```
char *szFruits[ ] = {
   "사과",
   "딸기",
   "포도",
   "오렌지",
   "자두"
};

CStringList list;  // 리스트 객체 생성
for(int i=0; i<5; i++)
   list.AddTail(szFruits[i]); // AddHead() 또는 AddTail()을 호출하여 원소를
                              // 리스트의 앞이나 뒤에 추가
```


리스트 클래스

- 순환

```
// 리스트 제일 앞에서 출발하여 순환한다
POSITION pos = list.GetHeadPosition(); // 리스트의 시작 위치를 얻고
while(pos != NULL){
   CString string = list.GetNext(pos);      // 차례로 데이터에 접근
   cout << (LPCTSTR)string << endl;
}
cout << endl;

// 리스트 제일 뒤에서 출발하여 순환한다
pos = list.GetTailPosition();               // 리스트의 끝 위치를 얻고
while(pos != NULL){
   CString string = list.GetPrev(pos);      // 차례로 데이터에 접근
   cout << (LPCTSTR)string << endl;
}
```


리스트 클래스

- 항목 삽입과 삭제

```
// POSITION 타입의 변수 pos는 이전의 예제에서 선언한 것이다.
pos = list.Find("포도");              // 데이터의 위치를 얻음
list.InsertBefore(pos, "살구");       // pos 위치 앞쪽에 데이터 삽입
list.InsertAfter(pos, "바나나");      // pos 위치 뒤쪽에 데이터 삽입
list.RemoveAt (pos);                  // pos 위치 데이터 삭제

// 항목 삽입과 삭제 후 결과를 확인한다.
pos = list.GetHeadPosition();
while(pos != NULL){
   CString string = list.GetNext(pos);
   cout << (LPCTSTR)string << endl;
}
```

리스트 클래스

- 템플릿 리스트 클래스

```
#include "stdafx.h"
#include "Console.h"
#include <afxtempl.h>

CWinApp theApp;

using namespace std;

struct Point3D {
    int x, y, z;
    Point3D() {}
    Point3D(int x0, int y0, int z0) { x = x0; y = y0; z = z0; }
};
```


리스트 클래스

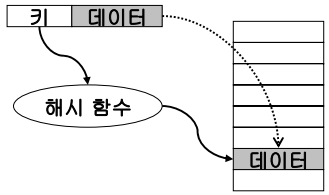
```
int _tmain(int argc, TCHAR* argv[ ], TCHAR* envp[ ])
{
    int nRetCode = 0;

    if (!AfxWinInit(...))
    {
        // 생략 ...
    }
    else
    {
        CList<Point3D, Point3D&> list;
        for(int i=0; i<5; i++)
            list.AddTail(Point3D(i, i*10, i*100));
        POSITION pos = list.GetHeadPosition();
```


리스트 클래스

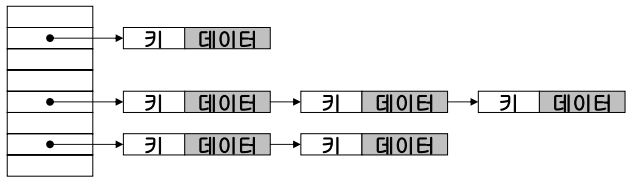
```
        while(pos != NULL) {
            Point3D pt = list.GetNext(pos);
            cout << pt.x << ", " << pt.y << ", " << pt.z << endl;
        }
    }

    return nRetCode;
}
```


맵 클래스

- 맵 동작 원리



- MFC 맵 클래스 구현



맵 클래스

- 템플릿 클래스
  - afxtempl.h 헤더 파일

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

맵 클래스

- 비 템플릿 클래스
  - afxcoll.h 헤더 파일

클래스 이름	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

- 생성, 초기화, 검색

```
CMapStringToString map;
map["사과"] = "Apple";              // "사과" 키에 "Apple" 데이터 추가
map["딸기"] = "Strawberry";
map["포도"] = "Grape";
map["우유"] = "Milk";

CString str;
if(map.Lookup("딸기", str))         // "딸기" 키로 데이터 검색
   cout << "딸기 -> " << (LPCTSTR)str << endl;
```


맵 클래스

- 순환

```
POSITION pos = map.GetStartPosition();      // 시작위치 얻기
while(pos != NULL){
   CString strKey, strValue;
   map.GetNextAssoc(pos, strKey, strValue); // 다음 키와 데이터 얻기
   cout << (LPCTSTR)strKey << " -> " <<
      (LPCTSTR)strValue << endl;
}
```

맵 클래스

- 삽입과 삭제

```
map.RemoveKey("우유");          // "우유" 키와 데이터를 삭제
map["수박"] = "Watermelon";     // "수박" 키와 "Watermelon" 데이터 추가

// 항목 삽입과 삭제 후 결과를 확인
// POSITION 타입의 변수 pos는 이전의 예제에서 선언한 것
pos = map.GetStartPosition();
while(pos != NULL){
    CString strKey, strValue;
    map.GetNextAssoc(pos, strKey, strValue);
    cout << (LPCTSTR)strKey << " -> " <<
        (LPCTSTR)strValue << endl;
}
```


맵 클래스

- 템플릿 맵 클래스

```
#include "stdafx.h"
#include "Console.h"
#include <afxtempl.h>

CWinApp theApp;

using namespace std;

UINT AFXAPI HashKey(CString& str)
{
    LPCTSTR key = (LPCTSTR) str;
    UINT nHash = 0;
    while(*key)
        nHash = (nHash<<5) + nHash + *key++;
    return nHash;
}
```


맵 클래스

```
int _tmain(int argc, TCHAR* argv[ ], TCHAR* envp[ ])
{
    int nRetCode = 0;

    if (!AfxWinInit(...))
    {
        // 생략 ...
    }
    else
    {
        CMap<CString, CString&, UINT, UINT&> map;
        map[CString ("사과")] = 10;
        map[CString ("딸기")] = 25;
        map[CString ("포도")] = 40;
        map[CString ("수박")] = 15;
```


맵 클래스

```
        UINT nCount;
        if(map.Lookup(CString("수박"), nCount))
            cout << "수박 " << nCount << "상자가 남아있습니다."
            << endl;
    }

    return nRetCode;
}
```