Chapter 3

클래스 : 함수를 포함할 수 있는 구조체의 확장이라고 생각하면 편하다. (이 책의 주요 개념)

생성자와 소멸자 : 클래스 객체가 생성 및 소멸할 때 '자동으로' 호출되는 함수

(클래스와 객체를 다루는 데 꼭 필요)

메서드: 함수 형태로 클래스의 실제 동작과 상태를 책임진다. (실제로 어떤 동작을 구현하는데 핵심)

객체지향 프로그래밍 개요

"객체란 변수들과 그와 관련된 메서드들이 모여서 이룬 하나의 꾸러미"

"클래스란 C의 구조체에서 확정된 변수, 객체, 함수를 포함한 하나의 틀"

```
/* 기존 절차지향 프로그래밍 코드 */
#include <stdio.h>

// 제작자의 코드
typedef struct USERDATA
{
    int nAge;
    char szName[32];
} USERDATA;

// 사용자의 코드
int main(void)
{
    USERDATA user = { 20, "철수" };
    printf("%d, %s\n", user.nAge, user.szName);

    return 0;
}
```

```
/* 제작자가 미리 구현한 코드 */
#include <stdio.h>

// 제작자의 코드
typedef struct USERDATA
{
    int nAge;
    char szName[32];
} USERDATA;

void PrintData(USERDATA* pUser)
{
    printf("%d, %s\n", pUser->nAge, pUser->szName);
}

// 사용자의 코드
int main(void)
{
    USERDATA user = { 20, "철수" };
    // printf("%d, %s\n",user.nAge, userszName);
    PrintData(&user);

    return 0;
}
```

```
/* 구조체와 함수 관계를 정의 */
#include <stdio.h>

// 제작자의 코드
typedef struct USERDATA
{
    int nAge;
    char szName[32];
    void(*Print)(struct USERDATA*);
} USERDATA;

void PrintData(USERDATA* pUser)
{
```

```
    printf("%d, %s\n", pUser->nAge, pUser->szName);
}

// 사용자의 코드
int main(void)
{
    USERDATA user = { 20, "철수",PrintData };
    // printf("%d, %s\n", user.nAge, user.szName);   // 1단계
    // PrintData(&user);                             // 2단계
    user.Print(&user);                               // 3단계

    return 0;
}
```

클래스

- 함수를 포함할 수 있는 구조체

```
/* 클래스를 이용해 객체지향 프로그램으로 변경 */
#include "pch.h"
#include <cstdio>

// 제작자의 코드
class USERDATA
{
public:
    // 멤버 변수 선언
    int nAge;
    char szName[32];

    // 멤버 함수 선언 및 정의
    void Print(void)
    {
        // nAge와 szName은 Print() 함수의 지역 변수가 아니다!
        printf("%d, %s\n", nAge, szName);
    }
};

// 사용자의 코드
int main()
{
    USERDATA user = { 10, "철수" };
    user.Print();

    return 0;
}
```

```
/* 멤버 변수 초기화를 위한 생성자 함수 사용 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CTest
{
public:
    // CTest 클래스의 '생성자 함수' 선언 및 정의
    CTest()
    {
        // 인스턴스가 생성되면 멤버 데이터를 '자동으로' 초기
        m_nData = 10;
    }

    // 멤버 데이터 선언
    int m_nData;

    // 멤버 함수 선언 및 정의
    void PrintData(void)
    {
        // 멤버 데이터에 접근하고 값을 출력한다.
        cout << m_nData << endl;
    }
};

// 사용자 코드
int main()
{
    CTest t;
    t.PrintData();

    return 0;
}
```

```
/* 생성자 함수의 역할 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CTest
```

```
/* 멤버 함수 선언과 정의를 분리 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CTest
```

```
{
public:
    // CTest 클래스의 '생성자 함수' 선언 및 정의
    CTest()
    {
        // 인스턴스가 생성되면 멤버 데이터를 '자동'으로 초기화한다.
        cout << "CTest() : 생성자 함수" << endl;
        m_nData = 10;
    }

    // 멤버 데이터 선언
    int m_nData;

    // 멤버 함수 선언 및 정의
    void PrintData(void)
    {
        // 멤버 데이터에 접근하고 값을 출력한다.
        cout << m_nData << endl;
    }
};

// 사용자 코드
int main()
{
    cout << "main() 함수 시작" << endl;

    CTest t;
    t.PrintData();

    cout << "main() 함수 끝" << endl;

    return 0;
}
```

```
{
public:
    // CTest 클래스의 '생성자 함수' 선언 및 정의
    CTest()
    {
        // 인스턴스가 생성되면 멤버 데이터를 '자동'으로 초기
        m_nData = 10;
    }
    // 멤버 데이터 선언
    int m_nData;

    // 멤버 함수 선언. 정의는 분리했다!
    void PrintData(void);
};

// 외부에 분리된 멤버 함수 정의
void CTest::PrintData(void)
{
    // 멤버 데이터에 접근하고 값을 출력한다.
    cout << m_nData << endl;
}

// 사용자 코드
int main()
{
    CTest t;
    t.PrintData();

    return 0;
}
```

```
/* 생성자 초기화 목록을 이용한 멤버 변수 초기화 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CTest
{
public:
    // 생성자 초기화 목록을 이용한 멤버 초기화
    CTest()
        : m_nData1(10),m_nData2(20)
    { }

    // 두 개의 멤버 데이터 선언
    int m_nData1;
    int m_nData2;

    // 멤버 함수 선언 및 정의
    void PrintData(void)
    {
        // 두 개의 멤버 데이터에 접근하고 값을 출력한다.
        cout << m_nData1 << endl;
        cout << m_nData2 << endl;
    }
};

// 사용자 코드
int main()
{
    CTest t;
```

```
/* C++11의 멤버 변수 초기화 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CTest
{
public:
    // 생성자 초기화 목록을 이용한 멤버 초기화
    CTest() { }

    // C++11부터 선언과 동시에 멤버 변수를 초기화할 수 있다
    int m_nData1 = 10;
    int m_nData2 = 20;

    void PrintData(void)
    {
        cout << m_nData1 << endl;
        cout << m_nData2 << endl;
    }
};

// 사용자 코드
int main()
{
    CTest t;
    t.PrintData();

    return 0;
}
```

```
    t.PrintData();

    return 0;
}
```

접근 제어 지시자

- 구조체가 클래스로 탈바꿈하도록 돕는 문법

지시자	설명
public	멤버에 관한 모든 외부 접근이 허용됩니다.
protected	멤버에 관한 모든 외부 접근이 차단됩니다. 단, 상속 관계에 있는 파생 클래스에서의 접근은 허용됩니다.
private	외부 접근뿐만 아니라 파생 클래스로부터의 접근까지 모두 차단됩니다. 클래스를 선언할 때 별도로 접근 제어 지시자를 기술하지 않으면 private으로 간주합니다.

기본은 private

```
/* 객체 내부 멤버 변수의 임의 접근 차단 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CMyData
{
    // 기본 접근 제어 지시자는 'private'이다.
    int m_nData;

public:
    int GetData(void) { return m_nData; }
    void SetData(int nParam) { m_nData = nParam; }
};

// 사용자 코드
int main()
{
    CMyData data;
    data.SetData(10);
    cout << data.GetData() << endl;

    return 0;
}
```

생성자와 소멸자

- 클래스 객체가 생성 및 소멸될 때 '자동으로' 호출되는 함수

클래스 - 다중 정의 가능 , 소멸자 - 다중 정의 불가능

```
/* 생성자와 소멸자 */
#include "pch.h"
#include <iostream>
using namespace std;

class CTest
{
```

```
public:
    CTest()
    {
        cout << "CTest::CTest()" << endl;
    }

    ~CTest()
    {
        cout << "~CTest::CTest()" << endl;
    }
};

int main()
{
    cout << "Begin" << endl;
    CTest a;
    cout << "End" << endl;

    return 0;
}
```

출력 : Begin → CTest::CTest( ) → End → ~CTest::CTest( )

- CTest a;를 전역 변수로 선언시 : CTest::CTest( ) → Begin → End → ~CTest::CTest( )

C++에서는 전역 변수가 main( )보다 먼저 호출된다.

- main( ) 함수가 호출되기 전에 생성자가 호출될 수 있다.
- 생성자는 다중 정의할 수 있다.
- 소멸자는 다중 정의할 수 없다.
- main( ) 함수가 끝난 후에 소멸자가 호출될 수 있다.
- 생성자와 소멸자는 생략할 수 있으나 생략할 경우 컴파일러가 만들어 넣는다.

```
/* 디폴트 생성자의 생략 */
#include "pch.h"
#include <iostream>
using namespace std;

class CTest
{
    int m_nData;

public:
    // 생성자의 매개변수로 전달된 값으로 멤버 변수를 초기화한다.
    CTest(int nParam) : m_nData(nParam)
    {
        cout << "CTest::CTest()" << endl;
    }

    ~CTest()
    {
        // 생성할 때 매개변수로 받은 값을 출력한다.
        cout << "~CTest::CTest() " << m_nData << endl;
    }
};

int main()
{
    cout << "Begin" << endl;
```

```
/* new와 delete 연산자 사용 */
#include "pch.h"
#include <iostream>
using namespace std;

class CTest
{
    int m_nData;

public:
    CTest()
    {
        cout << "CTest::CTest()" << endl;
    }

    ~CTest()
    {
        cout << "~CTest::CTest()" << endl;
    }
};

int main()
{
    cout << "Begin" << endl;

    // new 연산자를 이용해 동적으로 객체를 생성한다.
```

```
    CTest a(1);
    cout << "Before b" << endl;
    CTest b(2);
    cout << "End" << endl;

    return 0;
}
```

```
    CTest* pData = new CTest;
    cout << "Test" << endl;

    // delete 연산자를 이용해 객체를 삭제한다.
    delete pData;
    cout << "End" << endl;

    return 0;
}
```

배열로 생성시 배열의 수만큼 반복 (**배열 생성 = 배열 삭제**)

```
/* 참조자 선언을 위한 생성자 초기화 목록 이용 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CRefTest
{
public:
    // 참조형 멤버는 반드시 생성자 초기화 목록을 이용해 초기화한다.
    CRefTest(int& rParam) : m_nData(rParam) { };
    int GetData(void) { return m_nData; }

private:
    // 참조형 멤버는 객체가 생성될 때 반드시 초기화해야 한다.
    int& m_nData;
};

// 사용자 코드
int main()
{
    int a = 10;
    CRefTest t(a);

    cout << t.GetData() << endl;

    // 참조 원본인 a의 값이 수정되었다.
    a = 20;
    cout << t.GetData() << endl;

    return 0;
}
```

```
/* 생성자 다중 정의 */
#include "pch.h"
#include <iostream>
using namespace std;

class CMyData
{
public:
    // 디폴트 생성자는 없다.
    // 매개변수가 int 하나인 생성자 함수 선언 및 정의
    CMyData(int nParam) : m_nData(nParam) { };

    // 매개변수가 int 자료형 두 개인 생성자 함수 다중 정의
    CMyData(int x, int y) : m_nData(x + y) { };

    int GetData(void) { return m_nData; }

private:
    int m_nData;
};

int main()
{
    CMyData a(10);
    CMyData b(3, 4);

    cout << a.GetData() << endl;
    cout << b.GetData() << endl;

    return 0;
}
```

```
/* 다른 생성자를 추가로 부르는 생성자 초기화 함수 */
#include "pch.h"
#include <iostream>
using namespace std;

class CMyPoint
{
public:
    CMyPoint(int x)
    {
        cout << "CMyPoint(int)" << endl;
        // x 값이 100이 넘는지 검사하고 넘으면 100으로 맞춘다.
        if (x > 100)
            x = 100;

        m_x = 100;
```

```
/* 디폴트 생성자의 정의를 클래스 외부로 분리 */
#include "pch.h"
#include <iostream>
using namespace std;

class CTest
{
public:
    // 디폴트 생성자 선언
    CTest(void);
    int m_nData = 5;
};

// 클래스 외부에서 디폴트 생성자 정의
CTest::CTest(void) { }
```

```
    }

    CMyPoint(int x, int y)
        // x 값을 검사하는 코드는 이미 존재하므로 재사용한다.
        : CMyPoint(x)
    {
        cout << "CMyPoint(int, int)" << endl;

        // y 값이 200이 넘는지 검사하고 넘으면 200으로 맞춘다.
        if (y > 200)
            y = 200;

        m_y = 200;
    }

    void Print()
    {
        cout << "X:" << m_x << endl;
        cout << "Y:" << m_y << endl;
    }

private:
    int m_x = 0;
    int m_y = 0;
};

int main()
{
    // 매개변수가 하나인 생성자만 호출한다.
    CMyPoint ptBegin(110);
    ptBegin.Print();

    // 이번에는 두 생성자 모두 호출한다.
    CMyPoint ptEnd(50, 250);
    ptEnd.Print();

    return 0;
}
```

```
int main()
{
    CTest a;
    cout << a.m_nData << endl;

    return 0;
}
```

메서드

- 방법 - 클래스의 멤버 함수

this 포인터

- 작성 중인 클래스의 실제 인스턴스에 대한 주소를 가리키는 포인터

```
/* this 포인터 사용 */
#include "pch.h"
#include <iostream>
using namespace std;

class CMyData
{
public:
    CMyData(int nParam) : m_nData(nParam) { };
    void PrintData()
    {
        // m_nData의 값을 출력한다.
        cout << m_nData << endl;

        // CMyData 클래스의 멤버인 m_nData의 값을 출력한다.
        cout << CMyData::m_nData << endl;
```

```
        // 메서드를 호출한 인스턴스의 m_nData 멤버 값을 출력한다.
        cout << this->m_nData << endl;

        // 메서드를 호출한 인스턴스의 CMyData 클래스 멤버 m_nData를 출력한다.
        cout << this->CMyData::m_nData << endl;
    }

private:
    int m_nData;
};

int main()
{
    CMyData a(5), b(10);
    a.PrintData();
    b.PrintData();

    return 0;
}
```

CMyString

SetString( ) 메서드를 정의할 때 주의 사항

- 매개변수로 전달된 문자열의 길이를 측정하고 m_nLength에 저장
- 매개변수로 전달된 문자열이 저장될 수 있는 메모리를 동적 할당 (new연산자 사용)
- 동적 할당한 메모리에 문자열을 저장(m_pszData)
- 매개변수가 NULL이거나 문자열의 길이가 0인 경우를 고려
- 여기서 동적할당한 메모리는 언제 어디서 해제하는지 생각하고 대응
- 사용자가 다음 예와 같이 이 함수를 2회 호출하는 경우를 생각하고 대응

```
a.SetString("Hello");
a.SetString("World");
```

GetString( ) 메서드 정의

```
const char* CMyString::GetString(void)
{
  return m_pszData;
}
```

Release( ) 메서드를 정의할 때 주의 사항

- m_pszData라는 멤버 변수가 가리키는 메모리 해제 (delete연산자 사용)
- 사용자 코드에서 접근이 허용된 메서드들은 무엇이든 호출할 수 있으며 그 순서는 임의로 달라질 수 있다. 가령 Release( ) 메서드를 호출한 직후 SetString( ) 함수를 호출할 수 있는 것 이러한 상황을 고려하여 작성

상수형 메서드

- 멤버 변수에 읽기 접근은 가능하지만 쓰기는 허용되지 않는 메서드 - const

  (쓰기 - 각종 대입 연산자나 단항 연산자 등을 사용하는 것)

```
/* const 예약어로 선언한 상수형 메서드 */
#include "pch.h"
#include <iostream>
using namespace std;

class CTest
{
public:
    CTest(int nParam) : m_nData(nParam) { };
    ~CTest() { }

    // 상수형 메서드로 선언 및 정의했다.
    int GetData() const
    {
        // 멤버 변수의 값을 읽을 수는 있지만 쓸 수는 없다.
        return m_nData;
    }

    int SetData(int nParam) { m_nData = nParam; }

private:
    int m_nData = 0;
};

int main()
{
    CTest a(10);
    cout << a.GetData() << endl;

    return 0;
}
```

상수형 메서드는 절대로(혹은 문법적으로) 멤버 변수의 값을 쓸 수 없고, 상수형 메서드가 아닌 멤버는 호출할 수 없다.

“사용자 코드와 미래를 위해 const는 중요”

CMyString 코드들도 const 붙히기

상수형 메서드의 예외 사항

mutable, const_cast< > - const를 뽑아 쓸 수 있음

```
/* mutable 예약어 */
#include "pch.h"
#include <iostream>
using namespace std;

class CTest
{
public:
    CTest(int nParam) : m_nData(nParam) { };
    ~CTest() { }

    // 상수형 메서드로 선언 및 정의했다.
    int GetData() const
    {
```

```
/* const_cast< >를 사용한 상수형 참조 변경 */
#include "pch.h"
#include <iostream>
using namespace std;

void TestFunc(const int& nParam)
{
    // 상수형 참조였으나 일반 참조로 형변환했다.
    int& nNewParam = const_cast<int&>(nParam);

    // 따라서 l-value가 될 수 있다.
    nNewParam = 20;
}
```

```
        // 상수형 메서드라도 mutable 멤버 변수에는 값을 쓸 수 있다
        m_nData = 20;
        return m_nData;
    }

    int SetData(int nParam) { m_nData = nParam; }

private:
    mutable int m_nData = 0;
};

int main()
{
    CTest a(10);
    cout << a.GetData() << endl;

    return 0;
}
```

```
int main()
{
    int nData = 10;

    // 상수형 참조로 전달하지만 값이 변경된다.
    TestFunc(nData);

    // 변경된 값을 출력한다.
    cout << nData << endl;

    return 0;
}
```

멤버 함수 다중 정의

```
/* 서로 다은 자료형을 사용한 멤버 함수 */
#include "pch.h"
#include <iostream>
using namespace std;

void TestFunc(int nParam)
{
    cout << nParam << endl;
}

int main()
{
    TestFunc(10);
    TestFunc(5.5);

    return 0;
}
```

```
/* 멤버 함수 다중 정의 1 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CMyData
{
public:
    CMyData() : m_nData(0) { };

    int GetData(void) { return m_nData; }
    void SetData(int nParam) { m_nData = nParam; }

    // 매개변수가 double 자료형인 경우로 다중 정의했다.
    void SetData(double dParam) { m_nData = 0; }

private:
    int m_nData;
};

// 사용자 코드
```

```
/* delete 예약어를 사용해 컴파일 오류 발생 */
#include "pch.h"
#include <iostream>
using namespace std;

// 제작자 코드
class CMyData
{
public:
    CMyData() : m_nData(0) { };

    int GetData(void) { return m_nData; }
    void SetData(int nParam) { m_nData = nParam; }

    // 실수로 double 자료형 실인수가 넘어오는 경우를 차단한
    void SetData(double dParam) = delete;

private:
    int m_nData;
};

// 사용자 코드
```

```
int main()
{
    CMyData a;

    // CMyData::SetData(int) 메서드가 호출된다.
    a.SetData(10);
    cout << a.GetData() << endl;

    CMyData b;

    // CMyData::SetData(double) 메서드가 호출된다.
    b.SetData(5.5);
    cout << b.GetData() << endl;

    return 0;
}
```


```
int main()
{
    CMyData a;

    // CMyData::SetData(int) 메서드가 호출된다.
    a.SetData(10);
    cout << a.GetData() << endl;

    CMyData b;

    // CMyData::SetData(double) 메서드가 호출된다.
    b.SetData(5.5);
    cout << b.GetData() << endl;

    return 0;
}
```


정적 멤버 - static

- 클래스멤버로 들어온 전역 변수나 함수
- 인스턴스를 선언하지 않고 직접 호출
- this 포인터 X - 정적 변수는 반드시 선언과 정의를 분리

```
/* static 예약어를 사용한 정적 멤버 선언 및 정의 */
#include "pch.h"
#include <iostream>
using namespace std;

class CTest
{
public:
    CTest(int nParam) : m_nData(nParam) { m_nCount++; }
    int GetData() { return m_nData; }
    void ResetCount() { m_nCount = 0; }

    // 정적 메서드 선언 및 정의
    static int GetCount()
    {
        return m_nCount;
    };

private:
    int m_nData;

    // 정적 멤버 변수 선언(정의는 아니다!)
    static int m_nCount;
};

// CTest 클래스의 정적 멤버 변수 정의
int CTest::m_nCount = 0;

int main()
{
    CTest a(5), b(10);

    // 정적 멤버에 접근
    cout << a.GetCount() << endl;
    b.ResetCount();

    // 정적 멤버에 접근. 인스턴스 없이도 접근 가능!
    cout << CTest::GetCount() << endl;
```

```
    return 0;
}
```

문제

1. **private의 의미**

   접근을 못 하게 막는 것

2. **인스턴스가 생성될 때 자동으로 호출되는 함수 함수 원형의 가장 큰 외형상 특징**

   다중 정의 가능, 반환 형식이 없음

3. **다음 코드에서 ‘m_nData (nParam)’ 이 속한 부분의 명칭**

```
......
CTest(int nParam)
  : m_nData(nParam)
{
  ......
}
```

메서드 or 생성자 초기화 목록

4. **다음 코드에서 잘못된 점과 수정 방법**

```
class CRefTest
{
public:
  CRefTest(int &rParam)
  {
    m_nData = rParam;
  }

  int GetData(void) { return m_nData; }

private:
  int &m_nnData;
};
```

생성자 초기화 목록을 이용해 초기화해야한다

5. **메서드 함수 내부에서 실제 클래스 인스턴스의 주소를 가리키는 포인터의 이름**

   this

6. **상수형 메서드에서 할 수 없는 일**

   쓰기

7. **정적 멤버에서 사용할 수 없는 것**

   this 포인터