● 참조 멤버가 있는 클래스 생성자 CRefTest() 의 비교

```
// #1
    CRefTest(int &rParam) :m_nData(rParam)      // 생성자
    {
        cout << "CRefTest::CRefTest(int &rParam) " << m_nData << endl; // 생성자 호출 확인
    }
// #2
    CRefTest(int rParam) :m_nData(rParam)       // 생성자
    {
        cout << "CRefTest::CRefTest(int &rParam) " << m_nData << endl; // 생성자 호출 확인
    }
```

✓ 생성자의 매개변수를 참조형식(int &rParam)이 아니라 int rParam 이라고 작성하면 함수의 매개변수는 함수 내부의 자동변수와 같으므로 함수가 반환될 때 매개변수는 소멸

■ 생성자 다중정의

- 생성자도 함수이므로 당연히 다중정의할 수 있음.
- 생성자를 다중정의할 경우 제작자가 해야 할 일은 많아짐 → 사용자 코드는 더 간단하고 편리해질수 있음

```
#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  두개인 생성자 함수 다중 정의
    CMyData(int x, int y) : m_nData(x+y) { };

    int GetData() { return m_nData; }

private:
    int m_nData;
};

// 사용자 코드
int main()
{
    CMyData a(10);    // CMyData(int nParam) 호출
    CMyData b(3, 4);  // CMyData(int x, int y) 호출

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

    return 0;
}
```

✓ 어떤 생성자를 호출할 지는 컴파일러가 매개변수를 보고 알아서 결정

- 생성자가 다중 정의될 경우 사용자는 편하겠지만 제작자는 동일한 멤버를 초기화하는 코드를 여러 번 기술해야 하는 번거로움이 있음.
- C++ 표준부터는 '생성자 위임'이 지원 → 생성자 초기화 목록에서 다른 생성자를 추가로 부를 수 있음.

```
#include <iostream>
using namespace std;
class CMyPoint
{
public:
    CMyPoint(int x) {
        cout << " CMyPoint(int x) " << endl;
        // x값이 100이 넘는지 검사하고 넘으면 100으로 맞춘다.
        if (x > 100)
            x = 100;

        m_x = 100;
    }
    CMyPoint(int x, int y) : CMyPoint(x) // x값을 검사하는 코드는 이미 존재하므로 재사용한다.
    {
        cout << " CMyPoint(int, int) " << endl;

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

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

    return 0;
}
```

✓ CMyPoint(int x, int y) 생성자는 초기화 목록에서 CMyPoint(int) 생성자가 추가로 호출될 수 있도록 위임

✓ 같은 일을 하는 코드(x값을 검사하는 코드)가 여러번 반복해서 나타날 필요 없어짐.

■ 명시적 디폴트 생성자

- 함수 원형선언을 별도로 할 경우 선언과 정의가 분리된다.
- 클래스에서도 멤버함수의 선언과 정의를 별도로 분리할 수 있음.
- default 예약어를 사용하면 별도로 정의하지 않더라도 선언과 정의를 한 번에 끝낼 수 있음.

```
// 디폴트 생성자의 정의를 클래스 외부로 분리
#include <iostream>
using namespace std;

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

int main()
{
    CTest a;
    cout << a.m_nData << endl;
    return 0;
}
```

```
// 별도로 정의를 기술하지 않고도 선언과 정의가 완벽하게 분리됨
#include <iostream>
using namespace std;

class CTest
{
public:
    // 디폴트 생성자 선언 및 정의!
    CTest() = default;
    int m_nData = 5;
};

int main()
{
    CTest a;
    cout << a.m_nData << endl;
    return 0;
}
```

16. 메서드

- 메서드(Method)의 사전적 의미는 '방법'
- 클래스의 멤버함수를 메서드라고도 함 → 클래스가 제공하는 기능을 실행하는 방법
- 멤버함수의 원형

```
static 반환자료형 클래스이름::함수이름(매개변수) const;
```

- static과 const 예약어는 생략할 수 있음.
- static : 정적 멤버함수, const : 상수형 혹은 상수화된 멤버함수
- 메서드의 종류와 특징

종류	일반	상수화	정적	가상
관련예약어	-	const	ststic	virtual
this 포인터 접근	가능	가능	불가능	가능
일반 멤버 읽기	가능	가능	가능(제한적)	가능
일반 멤버 쓰기	가능	불가능	가능(제한적)	가능
정적 멤버 읽기	가능	가능	가능	가능
정적 멤버 쓰기	가능	불가능	가능	가능
특징	가장 보편적인 메서드	멤버 쓰기 방지가 목적	C의 전역함수와 유사	상속관계에서 의미가 큼

■ this 포인터

- 제작자가 작성중인 클래스의 실제 인스턴스에 대한 주소를 가리키는 포인터
- 18페이지 객체지향 프로그래밍의 개요에서 설명한 3단계를 보면 user.Print(&user); 객체가 선언된 후 인스턴스의 주소를 넘겨주도록 되어 있음.

```
// 사용자 코드
int main()
{
    USERDATA user = { 20, "철수", PrintData };
    // printf("%d %s\n", user.nAge, user.szName);   // 1단계
    // PrintData(&user);                            // 2단계
    user.Print(&user);                              // 3단계
    return 0;
}
```

- C++에서는 그 주소가 눈에 보이지 않을 뿐 실제로는 user.Print(&user);처럼 존재함.
- 매개변수로 전달된 인스턴스의 주소는 이름이 this 인 포인터변수(지역변수)로 저장됨.

- this 포인터 사용

```
#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_nData의 값을 출력한다.
        cout << this->CMyData::m_nData << endl;
    }
private:
    int m_nData;
};
int main()
{
    CMyData a(5), b(10);
    a.PrintData(); // &a는 보이지 않지만 실제로는 전달된다.
    b.PrintData(); // &b는 보이지 않지만 실제로는 전달된다.
    return 0;
}
```