■ 상수형 메서드

- 상수형 메서드 혹은 상수화된 메서드는 멤버 변수에 읽기 접근은 가능하지만 쓰기는 허용되지 않는 메서드
- '쓰기'란 각종 대입 연산자나 단항 연산자 등을 사용하는 것을 말함
- 선언방법은 함수 원형뒤에 형한정어인 const 예약어만 붙이면 됨!
- const 예약어로 선언한 상수형 메서드

```
#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 수 는 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;
}
```

- 상수화된 메서드에서 멤버함수라 하더라도 호출할 수 없습니다.(상수화된 메서드가 아닌 멤버함수)
  다음과 같은 코드는 허용되지 않음.(SetData(20) 함수가 상수화된 메서드가 아니므로!)

```
    // 상수형 메서드로 선언 및 정의했다.
    int GetData() const
    {
        SetData(20);

        // 멤버 변수의 값을 읽을 수 는 있지만 쓸 수는 없다.
        return m_nData;
    }
```

```
error C2662: 'int CTest::SetData(int)': 'this' 포인터를 'const CTest'에서 'CTest &'(으)로 변환할 수 없습니다.
```

- 상수형 메서드의 내부특징은 상수화 하는 방법이 'this 포인터를 상수형 포인터로 변경하는 것'
- 코드로 말하면 this 포인터의 형식이 CTest* this; 가 아니라 const CTest* this; 임!

- 다음과 같이 대입 연산자를 사용할 수도 없습니다.

```
    // 상수형 메서드로 선언 및 정의했다.
    int GetData() const
    {
        m_nData = 20;

        // 멤버 변수의 값을 읽을 수 는 있지만 쓸 수는 없다.
        return m_nData;
    }
```

```
error C3490: 'm_nData'은(는) const 개체를 통해 액세스되고 있으므로 수정할 수 없습니다.
```

- 상수형 메서드는 절대로(혹은 문법적으로) 멤버 변수의 값을 쓸 수 없고, 상수형 메서드가 아닌 멤버는 호출할 수 없다!
- 코드로 말하면 this 포인터의 형식이 CTest* this; 가 아니라 const CTest* this; 임!

■ 상수형 메서드의 예외사항

- 형한정어인 const 예약어를 계단에 설치한 난간에 비유할 때, '미래' 어느 시점에 난간을 제거해야할 수도 있음.
- mutable 예약어, C++ 전용 형변환 연산자 const_cast<>
- 상수화된 대상에서 const 예약어가 지정된 멤버를 뽑아 값을 쓰거나 호출할 수 있음.

```
// mutable 예약어 사용
#include <iostream>
using namespace std;
class CTest
{
public:
    CTest(int nParam) {}
    ~CTest() { }

    int GetData() const
    {
        // 상수형 메서드라도 mutable 멤버변수에는 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;
}
```

- mutable로 선언한 멤버 변수의 값은 상수형 메서드에서도 쓰기가 허용

● const_cast<>을 사용한 상수형 참조 변경

```
#include <iostream>
using namespace std;

void TestFunc(const int& nParam)
{
    int& nNewParam = const_cast<int&>(nParam);

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

int main()
{
    int nData = 10;

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

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

    return 0;
}
```

✓ TestFunc() 함수의 매객변수의 형식은 const int & 임(읽기만 가능)
✓ 이것을 int & 로 변경 (쓰기도 가능하도록)
✓ 사용한 형변환 연산자 const_cast
const_cast<새형식>(대상) : 대상을 괄호로 묶어야 합니다. 그렇지 않으면 오류 메시지 발생

17. 정적 멤버

- 변수든 함수든 '정적 멤버'는 사실상 전역변수나 함수와 같다. 단지 클래스의 멤버로 들어왔을 뿐..
- 정적 멤버를 선언하려면 맨 앞에 static 예약어를 작성
- 정적 멤버함수는 인스턴스를 선언하지 않고 직접 호출할 수 있음.
- this 포인터를 사용할 수 없으며 정적변수는 반드시 선언과 정의를 분리해야 함.

```
#include <iostream>
using namespace std;
class CTest
{
public:
    CTest(int nParam) : m_nData(nParam) { m_nCount++; }
    ~CTest() { }

    int GetData() const { return m_nData; }
    void ResetCount() { m_nCount = 0;  }

    // 정적 메서드 정의 및 정의
    static int GetCount() {
        return m_nCount;
    }
private:
    int m_nData = 0;

    // 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(10), b(5);

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

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

    return 0;
}
```