19. 묵시적 변환

■ 변환 생성자

- 매개변수가 한 개인 생성자를 다른말로 변환 생성자(Conversion constructor) 라고도 함
- 문제는 변환생성자가 은근슬쩍 호출되는 경우가 있음
- 더 큰 문제는 불필요한 임시 객체를 만들어냄으로써 프로그램의 효율을 갉아먹는 원인이 된다는 것!

```
#include <iostream>
using namespace std;
// 제작자 코드
class CTestData
{
public:
    // 매개변수가 하나뿐인 생성자는 형변환이 가능하다.
    CTestData(int nParam) :m_nData(nParam)
    {
        cout << " CTestData(int) " << endl;
    }
    CTestData(const CTestData& rhs) :m_nData(rhs.m_nData)
    {
        cout << " CTestData(const CTestData &)" << endl;
    }
    int GetData() const { return m_nData;  }
    void SetData(int nParam) { m_nData = nParam; }
private:
    int m_nData;
};
// 사용자 코드
// 매개변수가 클래스 형식이면 변환 생성이 가능하다.
void TestFunc(CTestData param)
{
    cout << "  TestFunc(): " << param.GetData() << endl;
}
int main()
{
    // int 자료형에서 CTestData 형식을 변환될 수 있다.
    TestFunc(5);

    return 0;
}
```

실행결과

```
 CTestData(int)
 TestFunc(): 5
계속하려면 아무 키나 누르십시오 . . .
```

- CTestData 클래스는 int 자료형에 대한 변환 생성자를 제공
- 함수의 매개변수 형식으로 CTestData 클래스가 사용된다면 호출자 코드의 형식이 CTestData가 아니라 int 자료형이 될 수 있음.
- TestFunc() 함수의 매개변수는 CTestData 클래스 형식임을 확인
- CTestData 클래스는 int 자료형에 대한 변환생성자 CTestData(int)를 제공하기에 TestFunc(5); 코드가 작성할 수 있음.

- 소멸자 추가, TestFunc() 함수의 매개변수를 CTestData param에서 const CTest &param으로 수정

```
#include <iostream>
using namespace std;
// 제작자 코드
class CTestData
{
public:
    // 매개변수가 하나뿐인 생성자는 형변환이 가능하다.
    CTestData(int nParam) : m_nData(nParam)
    {
        cout << " CTestData(int) " << endl;
    }
    CTestData(const CTestData& rhs) : m_nData(rhs.m_nData)
    {
        cout << " CTestData(const CTestData &)" << endl;
    }
    ~CTestData() // 소멸자 추가
    {
        cout << " ~CTestData() " << endl;
    }
    int GetData() const { return m_nData;  }
    void SetData(int nParam) { m_nData = nParam; }
private:
    int m_nData;
};
// 사용자 코드
// 매개변수가 클래스에 대한 참조 형식이며 묵시적으로 변환 생성된다.
void TestFunc(const CTestData & param)
{
    cout << " TestFunc(): " << param.GetData() << endl;
}
int main()
{
    cout << " *****Begin***** " << endl;
    // 새로운 CTestData 객체를 생성하고 참조로 전달한다.
    TestFunc(5);

    // 함수가 반환되면서 임시 객체는 소멸한다.
    cout << " ***** End ***** " << endl;
    return 0;
}
```

실행결과

```
*****Begin*****
CTestData(int)
TestFunc(): 5
~CTestData()
***** End *****
```

- 실행결과를 보면 분명히 변환생성자가 호출됐음을 확인할수 있음
- TestFunc(const CTestData & param) 함수 매개변수로 CTestData 객체를 넘겨줘야 하는데
- main() 함수에 CTestData클래스 객체를 선언했거나 동적으로 생성하는 코드가 없음.
- 컴파일러가 '알아서' 임시 객체를 생성한 후 이 임시 객체에 대한 참조가 TestFunc() 함수로 전달
- 임시객체는 TestFunc() 함수를 반환함과 동시에 소멸
- TestFunc(5); 라는 코드가 실제로는 TestFunc(CTestData(5)); 라고 코딩한것과 같아짐.
- 클래스를 매개변수로 사용할 거라면 무조건 참조형식으로 사용하는 것에 덧붙여 묵시적 변환 생성자를 지원하는 클래스인지 꼭 확인해야 함!

- 묵시적 변환 생성자가 사용자 모르게 호출될 가능성을 차단!!(변환 생성자 앞에 explicit 예약어)

```
#include <iostream>
using namespace std;
// 제작자 코드
class CTestData
{
public:
    // 매개변수가 하나뿐인 생성자는 형변환이 가능하다.
    // 하지만 묵시적으로는 불가능하도록 차단한다.
    explicit CTestData(int nParam) : m_nData(nParam)
    {
        cout << " CTestData(int) " << endl;
    }
    CTestData(const CTestData& rhs) : m_nData(rhs.m_nData)
    {
        cout << " CTestData(const CTestData &)" << endl;
    }
    ~CTestData() // 소멸자 추가
    {
        cout << " ~CTestData() " << endl;
    }
    int GetData() const { return m_nData;  }
    void SetData(int nParam) { m_nData = nParam; }
private:
    int m_nData;
};
// 사용자 코드
// 매개변수가 클래스에 대한 참조 형식이며 묵시적으로 변환 생성된다.
void TestFunc(const CTestData & param)
{
    cout << " TestFunc(): " << param.GetData() << endl;
}
int main()
{
    cout << " *****Begin***** " << endl;

    TestFunc(5); // 컴파일 오류가 발생한다!

    cout << " ***** End ***** " << endl;
    return 0;
}
```

실행결과

```
error C2664: 'void TestFunc(const CTestData &)': 인수 1을(를) 'int'에서 'const CTestData &'(으)로 변환할 수 없습니다.
```

- explicit 예약어를 적용할 경우 사용자 코드에서 묵시적으로 변환이 일어나지 않게끔 하여 임시 객체가 생성되지 못하게 함.
- TestFunc(5); 코드는 허용 안됨
- 무조건 임시객체가 생성된다는 사실이 명확히 드러나도록 TestFunc(CTestData(5)); 형태는 가능!

■ 허용되는 변환

- 클래스가 변환생성자를 제공하면 두 형식 사이에 호환성이 생김
- CTestData(int); 변환생성자를 제공했다면 int 자료형이 CTestData 클래스 형식으로 변할 수 있음
- 그러나 CTestData 클래스 형식이 int 자료형으로 변환될 수는 없음(반쪽자리 변환)
- 형변환 연산자(혹은 형변환자)를 만들어 넣으면 CTestData 클래스 형식이 int 자료형으로 변환가능!

```
#include <iostream>
using namespace std;
// 제작자 코드
class CTestData
{
public:
    explicit CTestData(int nParam) : m_nData(nParam) { }

    // CTestData 클래스는 int 자료형으로 변환될 수 있다!
    operator int() { return m_nData; }

    int GetData() const { return m_nData; }
    void SetData(int nParam) { m_nData = nParam; }
private:
    int m_nData = 0;
};
int main()
{
    CTestData a(10);
    cout << a.GetData() << endl;

    // CTestData 형식에서 int 자료형을 변환될 수 있다.
    cout << a << endl;
    cout << (int)a << endl;
    cout << static_cast<int>(a) << endl;

    return 0;
}
```

- 형변환 연산자 때문에 cout << a << endl; 코드 실행이 가능
- cout 객체는 CTestData 클래스 인스턴스를 화면에 출력할 수 없지만 형변환 연산자 덕분에 int 자료형으로 인식될수 있음.
- 클래스의 형변환 연산자에도 변환생성자처럼 explicit 예약어를 적용할 수 있음. a가 int 자료형으로 '묵시적으로' 변환을 방지.