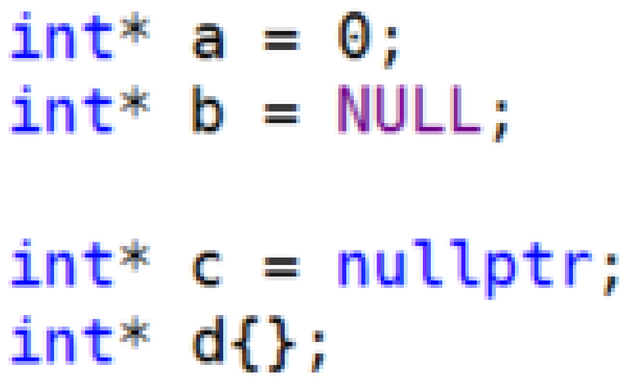
Modern C++

SSL

- C++11 이후 포인터 초기화

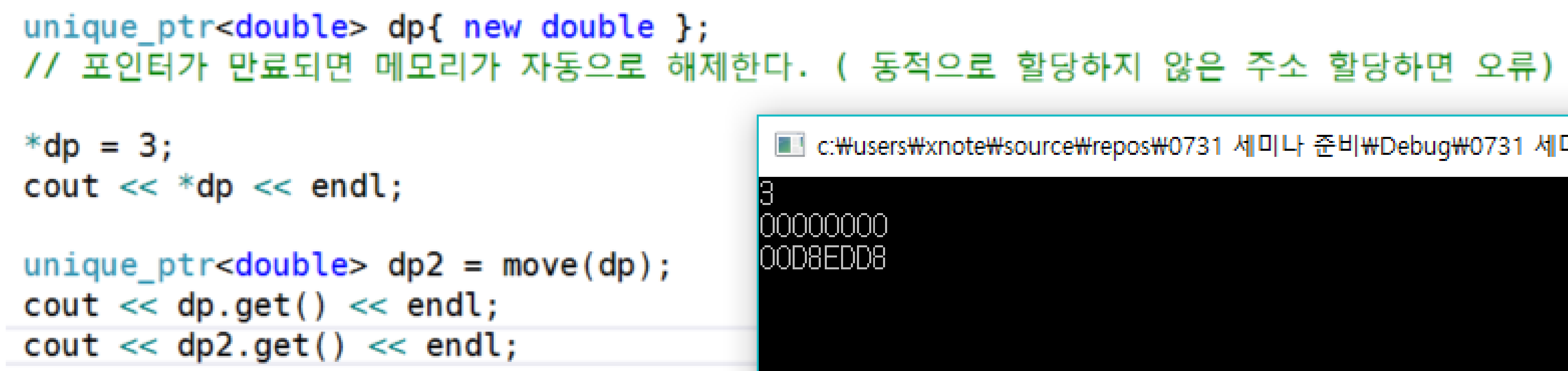


```
int* a = 0;
int* b = NULL;

int* c = nullptr;
int* d{};
```

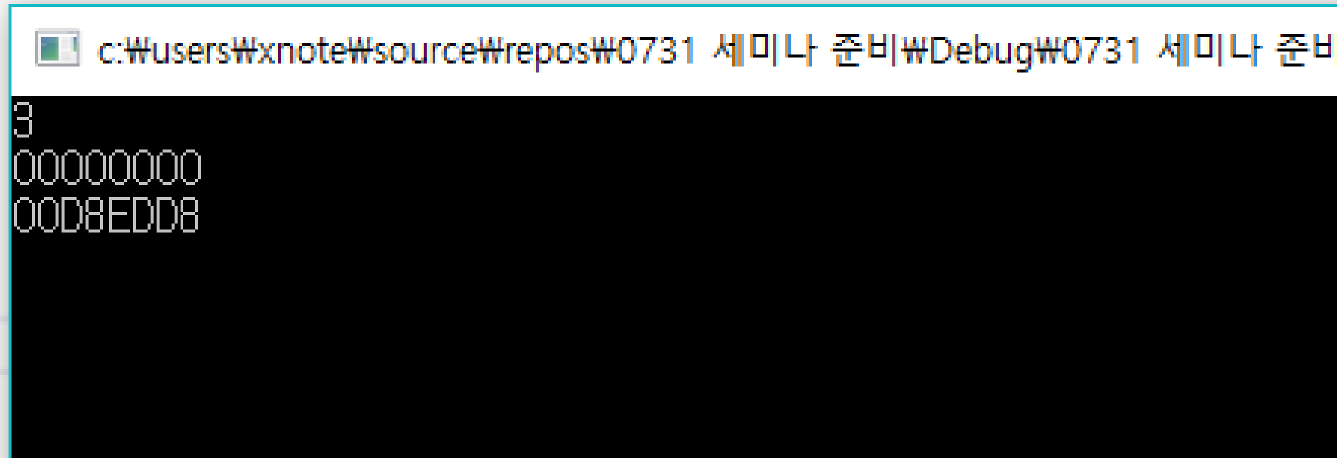
SSL

1. 스마트 포인터

- unique_ptr



```
unique_ptr<double> dp{ new double };
// 포인터가 만료되면 메모리가 자동으로 해제한다. ( 동적으로 할당하지 않은 주소 할당하면 오류)

*dp = 3;
cout << *dp << endl;

unique_ptr<double> dp2 = move(dp);
cout << dp.get() << endl;
cout << dp2.get() << endl;
```

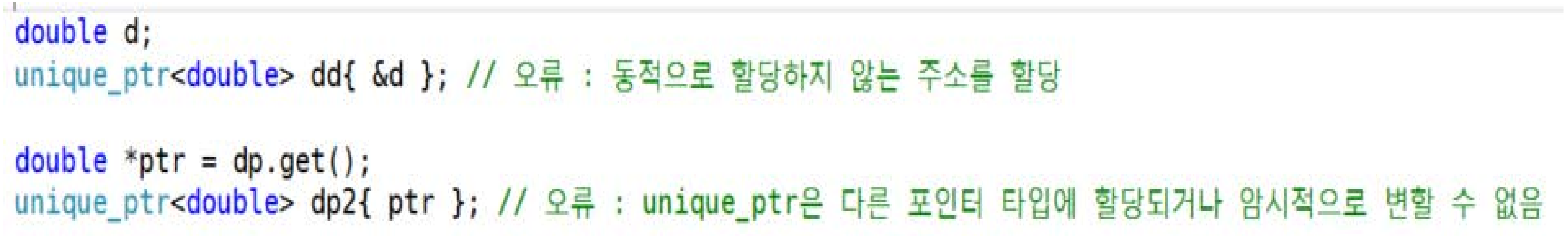


c:₩users₩xnote₩source₩repos₩0731 세미나 준비₩Debug₩0731 세미나 준비

```
3
00000000
00D8EDD8
```

SSL

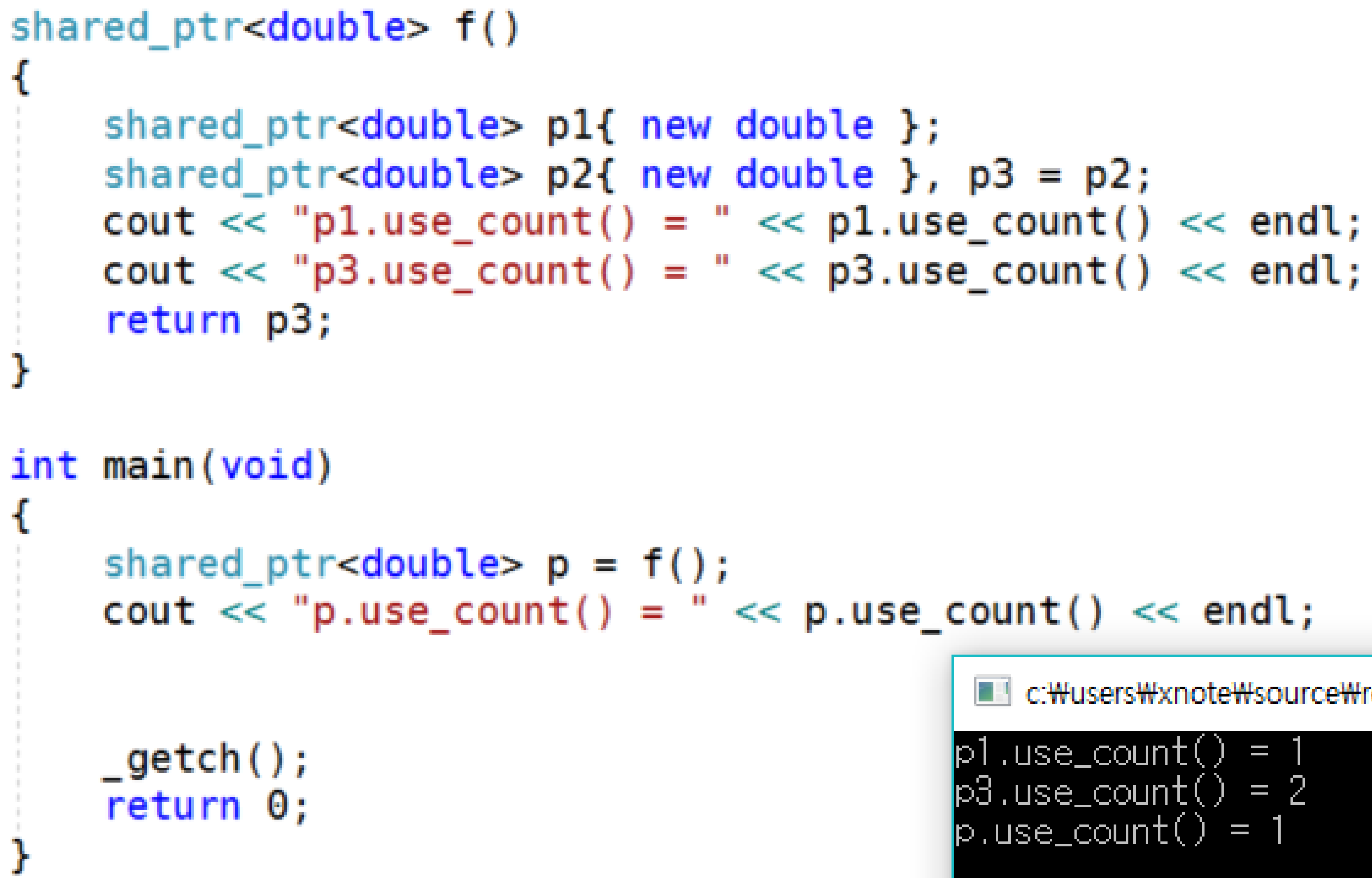
- unique_ptr 오류



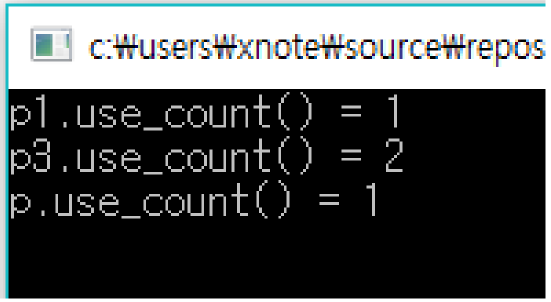
```
double d;
unique_ptr<double> dd{ &d }; // 오류 : 동적으로 할당하지 않는 주소를 할당

double *ptr = dp.get();
unique_ptr<double> dp2{ ptr }; // 오류 : unique_ptr은 다른 포인터 타입에 할당되거나 암시적으로 변할 수 없음
```

SSL

- shared_ptr



```
shared_ptr<double> f()
{
    shared_ptr<double> p1{ new double };
    shared_ptr<double> p2{ new double }, p3 = p2;
    cout << "p1.use_count() = " << p1.use_count() << endl;
    cout << "p3.use_count() = " << p3.use_count() << endl;
    return p3;
}

int main(void)
{
    shared_ptr<double> p = f();
    cout << "p.use_count() = " << p.use_count() << endl;


    _getch();
    return 0;
}
```



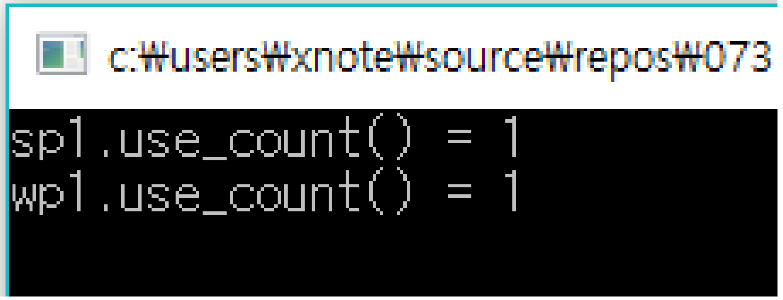
c:₩users₩xnote₩source₩repos

```
p1.use_count() = 1
p3.use_count() = 2
p.use_count() = 1
```

SSL

- weak_ptr

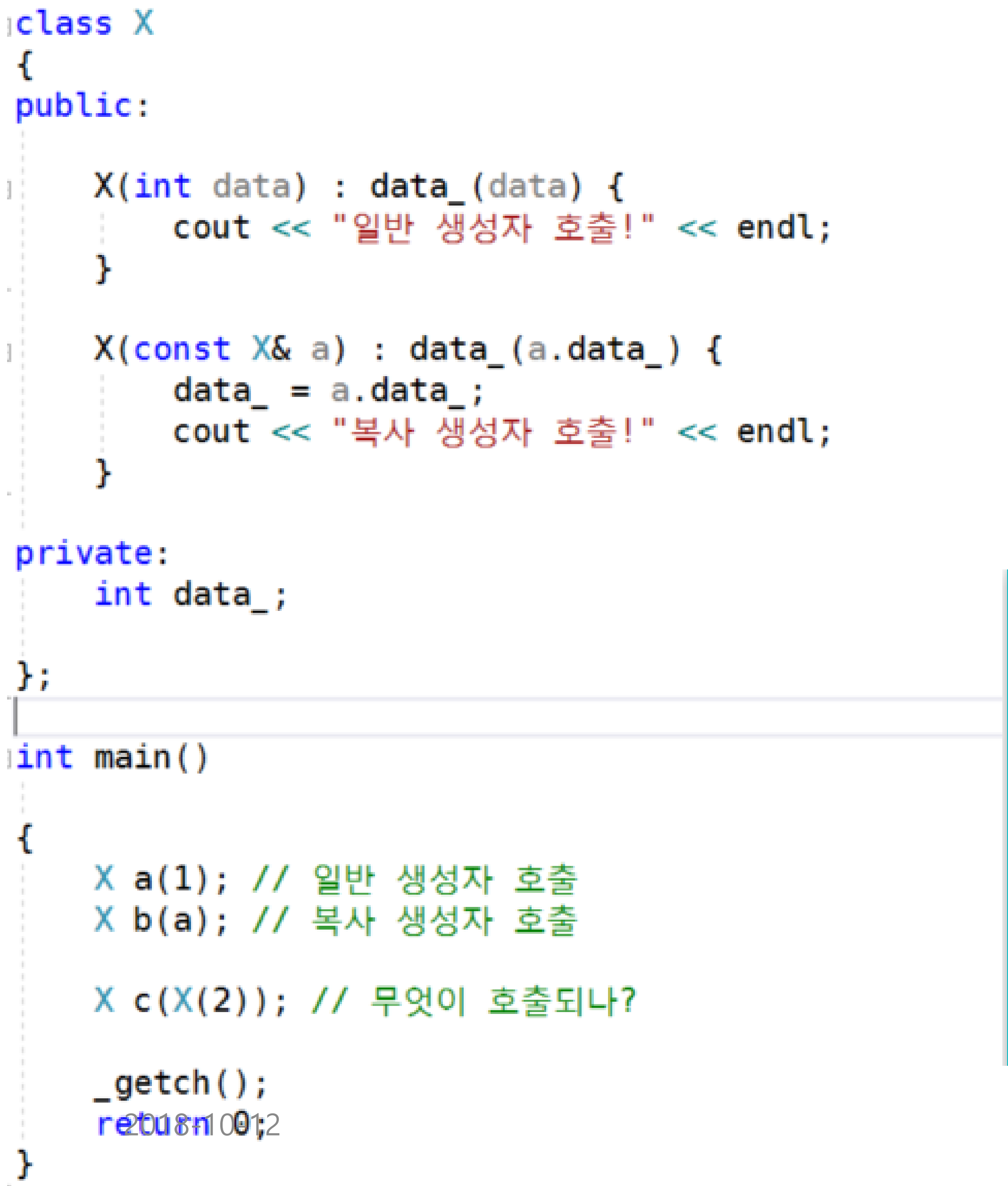
```cpp
shared_ptr<int> sp1(new int(10));
weak_ptr<int> wp1 = sp1;

cout << "sp1.use_count() = " << sp1.use_count() << endl;
cout << "wp1.use_count() = " << wp1.use_count() << endl;


_getch();
return 0;
```



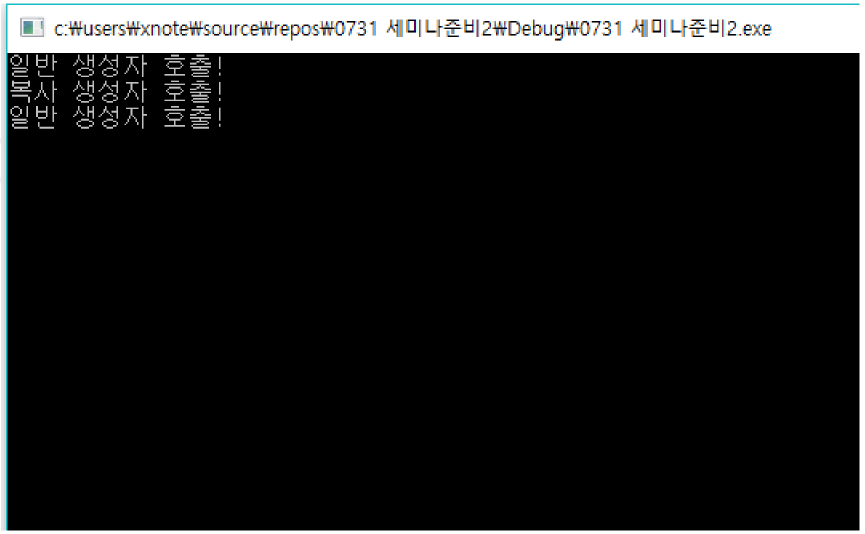
```
c:₩users₩xnote₩source₩repos₩073
sp1.use_count() = 1
wp1.use_count() = 1
```

SSL

2. 복사생략

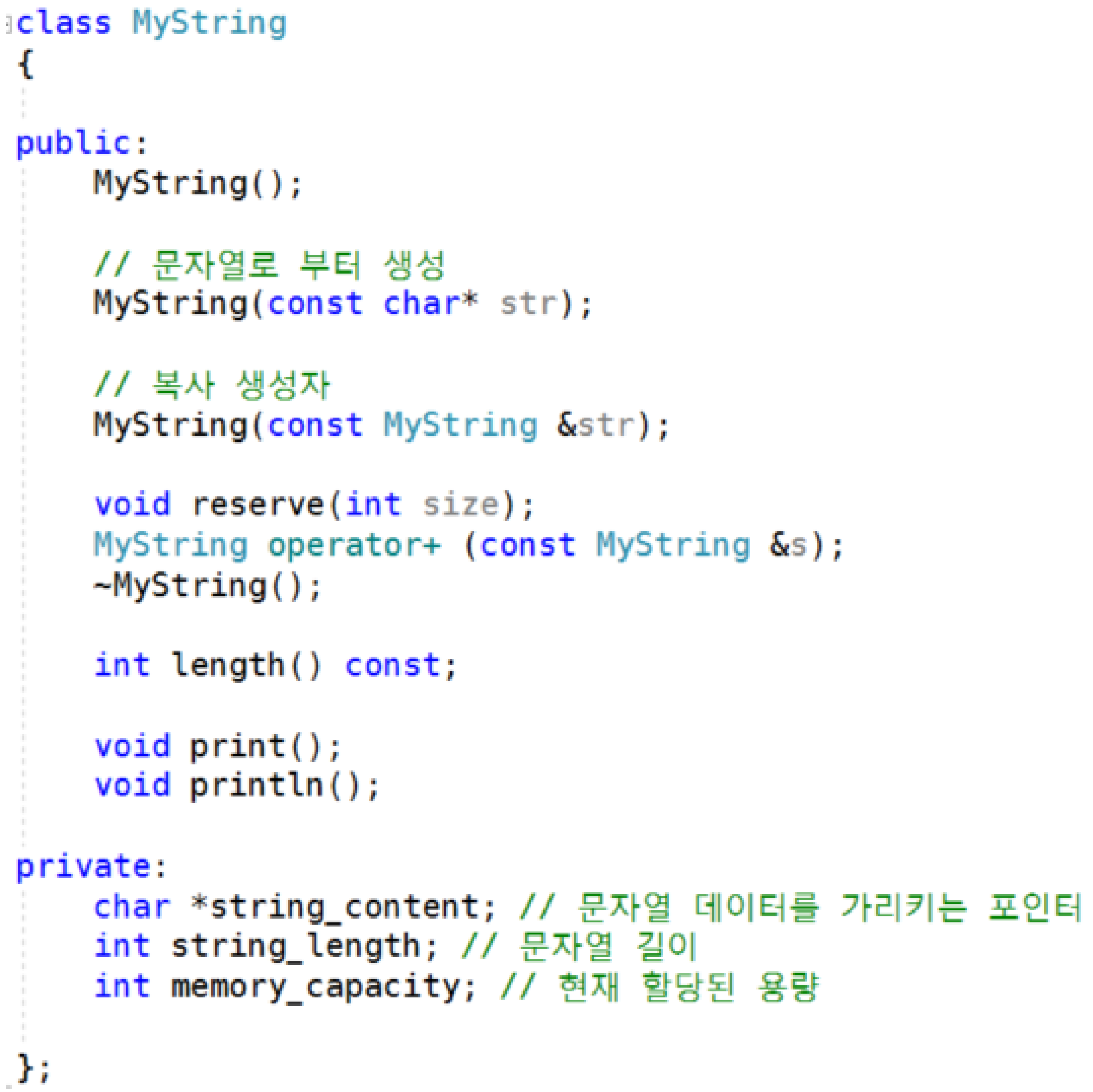


```cpp
class X
{
public:

    X(int data) : data_(data) {
        cout << "일반 생성자 호출!" << endl;
    }

    X(const X& a) : data_(a.data_) {
        data_ = a.data_;
        cout << "복사 생성자 호출!" << endl;
    }

private:
    int data_;

};

int main()

{
    X a(1); // 일반 생성자 호출
    X b(a); // 복사 생성자 호출

    X c(X(2)); // 무엇이 호출되나?

    _getch();
    return 0;
}
```

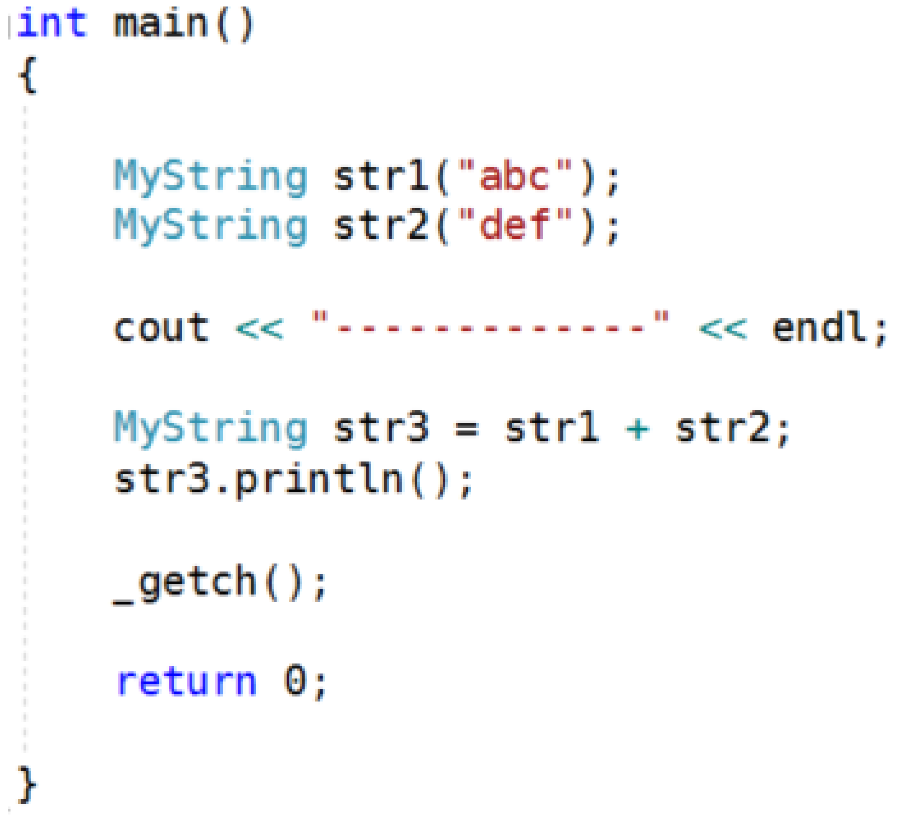


```
c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe
일반 생성자 호출!
복사 생성자 호출!
일반 생성자 호출!
```

SSL

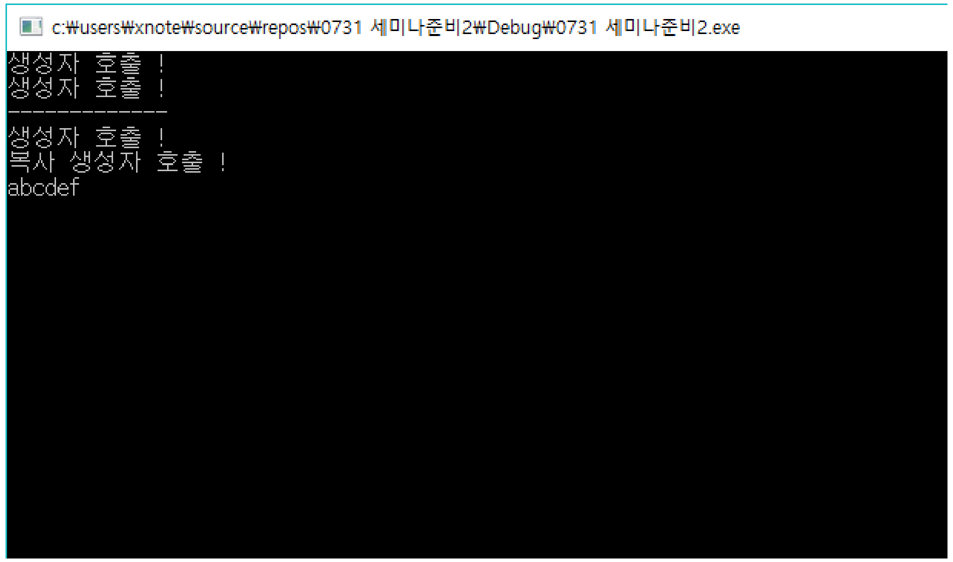


```cpp
class MyString
{

public:
    MyString();

    // 문자열로 부터 생성
    MyString(const char* str);

    // 복사 생성자
    MyString(const MyString &str);

    void reserve(int size);
    MyString operator+ (const MyString &s);
    ~MyString();

    int length() const;

    void print();
    void println();

private:
    char *string_content; // 문자열 데이터를 가리키는 포인터
    int string_length; // 문자열 길이
    int memory_capacity; // 현재 할당된 용량

};
```



```cpp
int main()
{

    MyString str1("abc");
    MyString str2("def");

    cout << "-------------" << endl;

    MyString str3 = str1 + str2;
    str3.println();

    _getch();

    return 0;

}
```

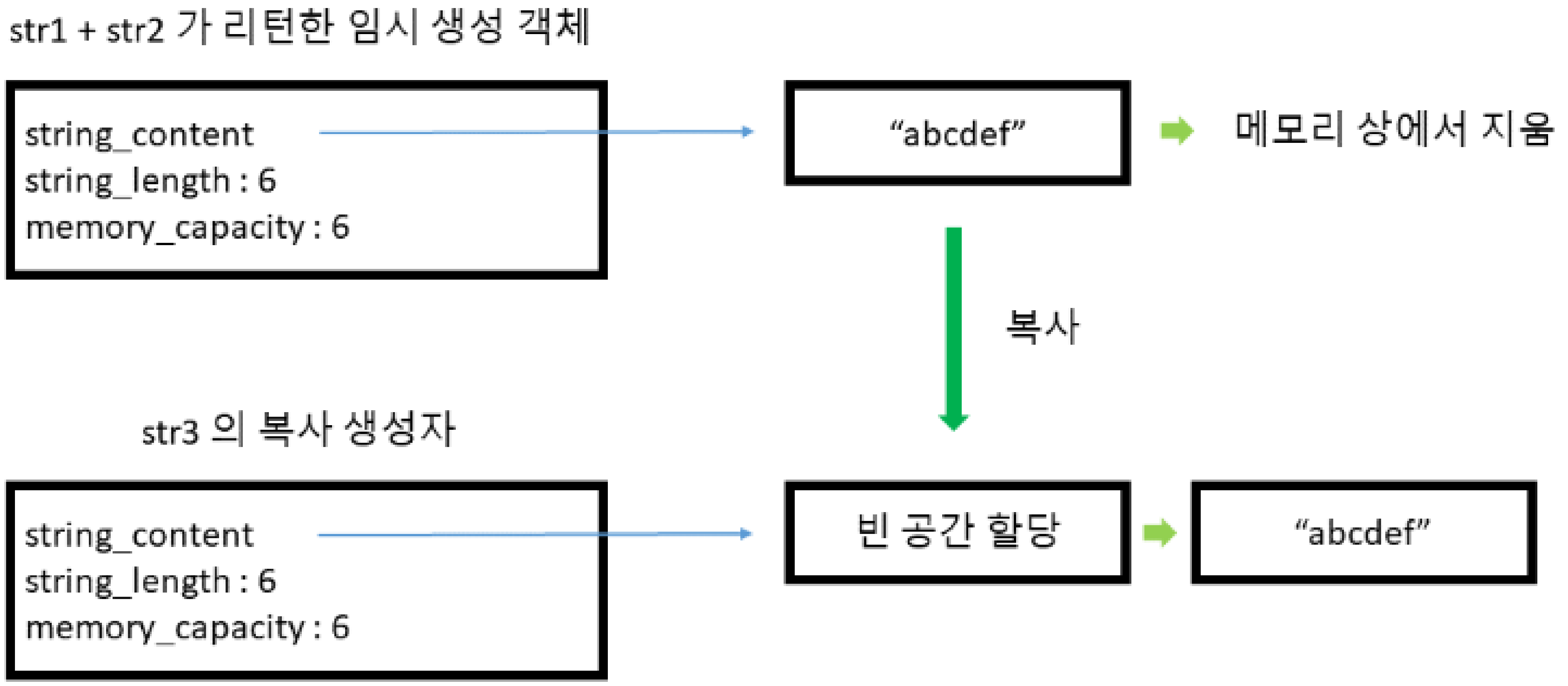
```cpp
MyString MyString::operator+ (const MyString &s)
{

    MyString str;
    str.reserve(string_length + s.string_length);
    for (int i = 0; i < string_length; i++)
        str.string_content[i] = string_content[i];
    for (int i = 0; i < s.string_length; i++)
        str.string_content[string_length + i] = s.string_content[i];

    str.string_length = string_length + s.string_length;

    return str;

}
```



c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe

```
생성자 호출 !
생성자 호출 !
-------------
생성자 호출 !
복사 생성자 호출 !
abcdef
```

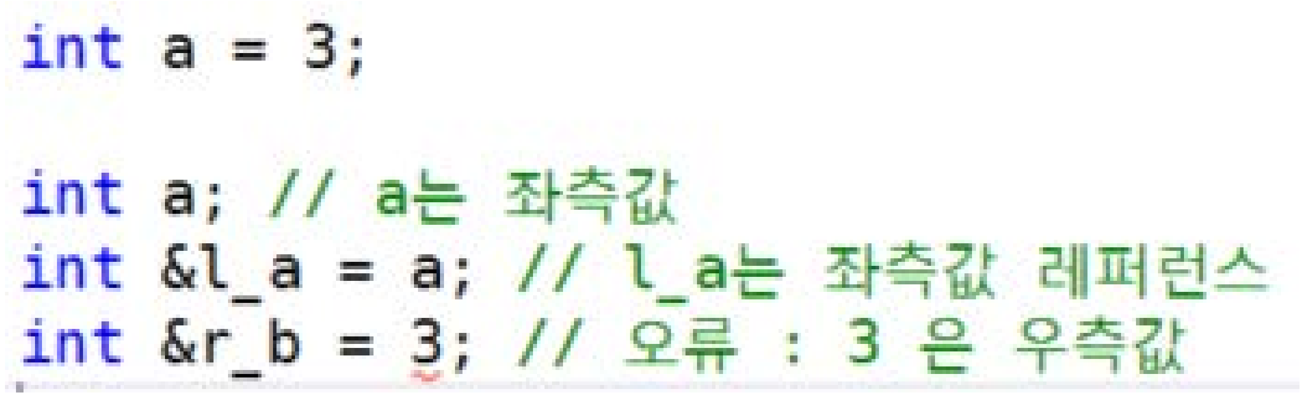
SSL


str1 + str2 가 리턴한 임시 생성 객체
string_content
string_length : 6
memory_capacity : 6
"abcdef"
메모리 상에서 지움
복사
str3 의 복사 생성자
string_content
string_length : 6
memory_capacity : 6
빈 공간 할당
"abcdef"

SSL

3. 우측값 레퍼런스

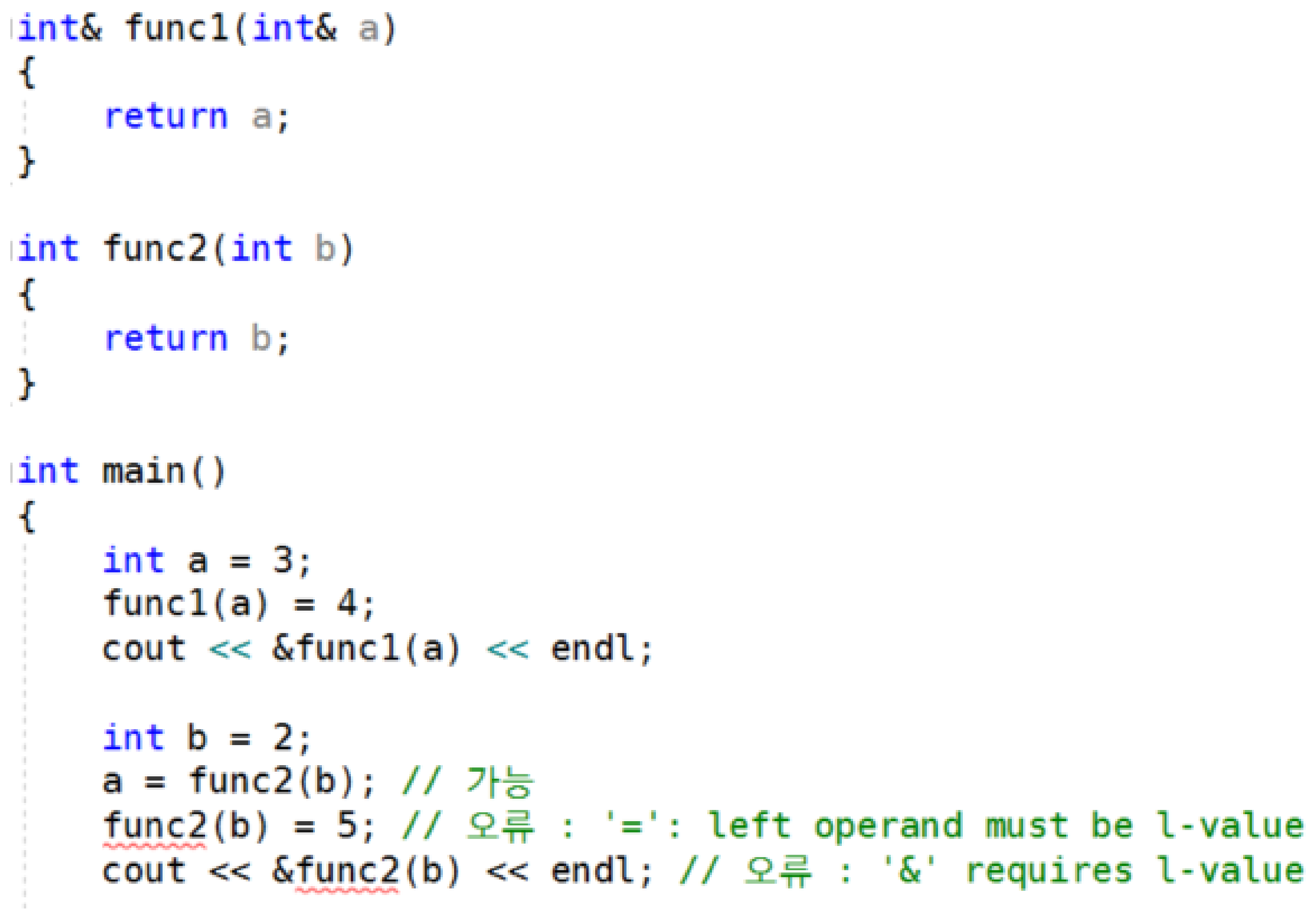
좌측값과 우측값



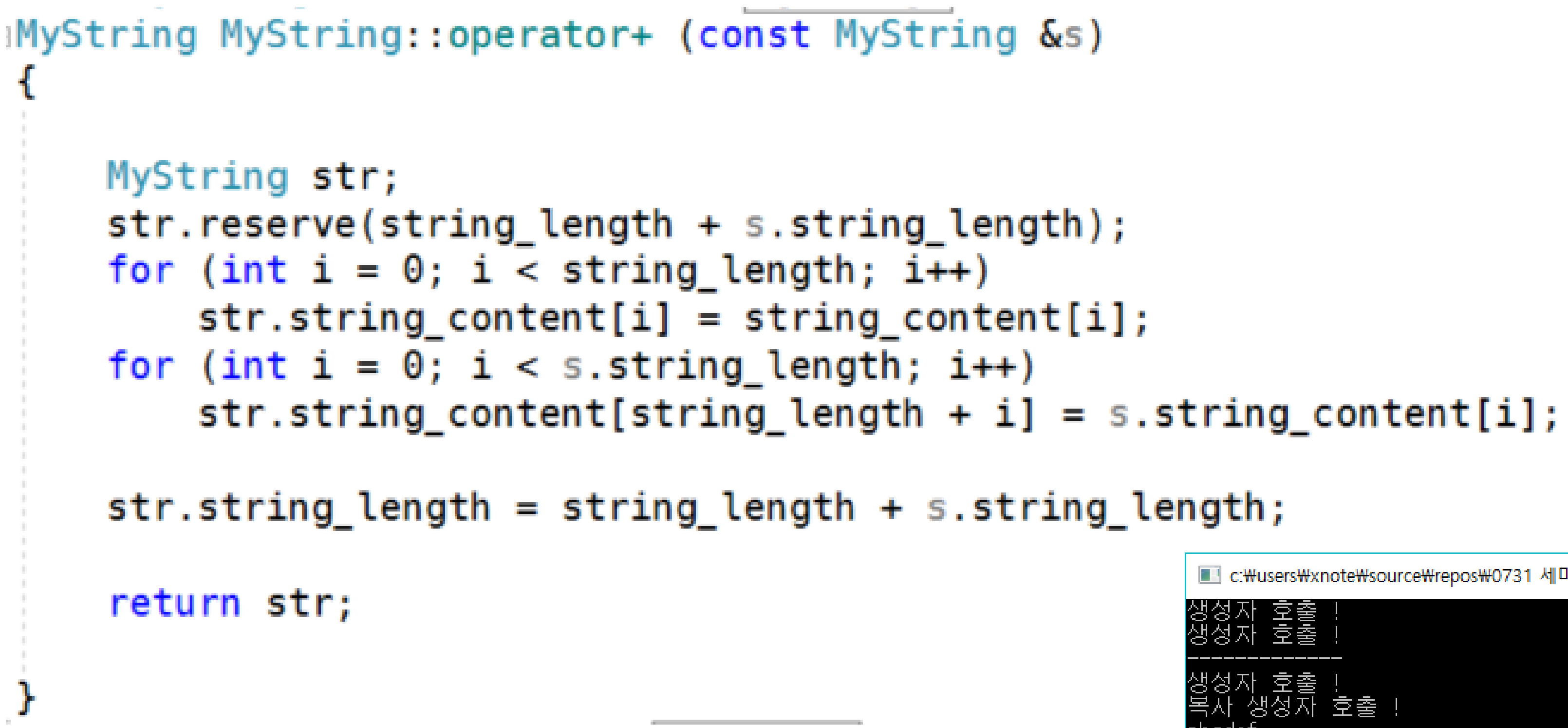
```
int a = 3;

int a; // a는 좌측값
int &l_a = a; // l_a는 좌측값 레퍼런스
int &r_b = 3; // 오류 : 3 은 우측값
```

& : 좌측값 레퍼런스(lvalue reference)
&& : 우측값 레퍼런스(rvalue reference)

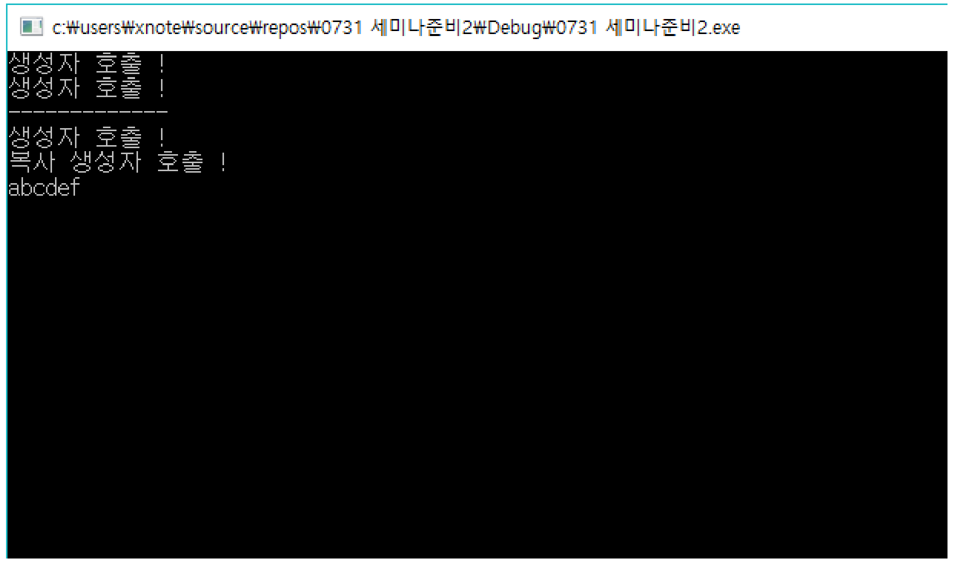
SSL



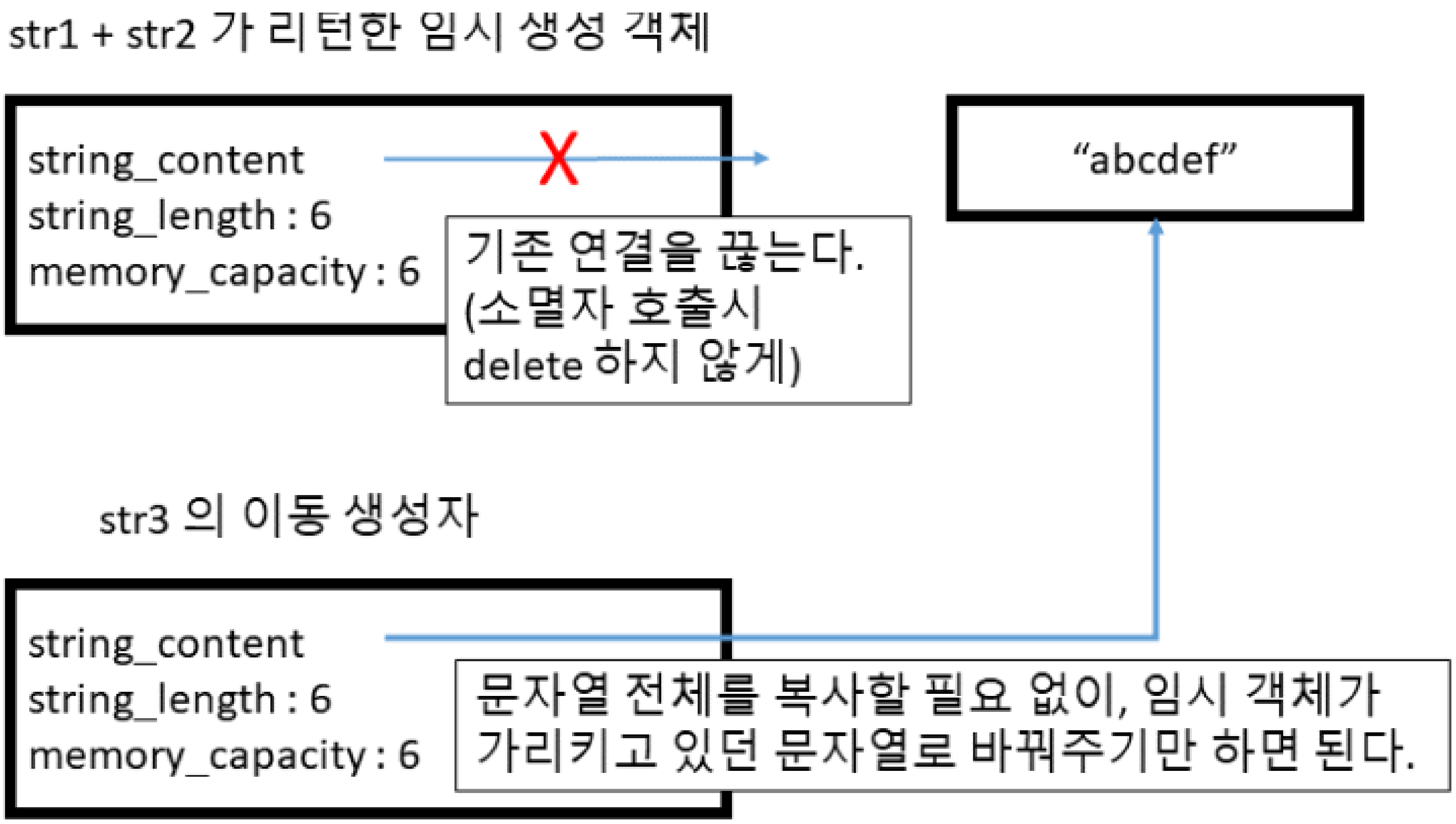
```
int& func1(int& a)
{
    return a;
}

int func2(int b)
{
    return b;
}

int main()
{
    int a = 3;
    func1(a) = 4;
    cout << &func1(a) << endl;

    int b = 2;
    a = func2(b); // 가능
    func2(b) = 5; // 오류 : '=': left operand must be l-value
    cout << &func2(b) << endl; // 오류 : '&' requires l-value
```



```cpp
MyString MyString::operator+ (const MyString &s)
{

    MyString str;
    str.reserve(string_length + s.string_length);
    for (int i = 0; i < string_length; i++)
        str.string_content[i] = string_content[i];
    for (int i = 0; i < s.string_length; i++)
        str.string_content[string_length + i] = s.string_content[i];

    str.string_length = string_length + s.string_length;

    return str;

}
```


c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe

```
생성자 호출 !
생성자 호출 !
-------------
생성자 호출 !
복사 생성자 호출 !
abcdef
```



기존 연결을 끊는다.
(소멸자 호출시
delete 하지 않게)

문자열 전체를 복사할 필요 없이, 임시 객체가
가리키고 있던 문자열로 바꿔주기만 하면 된다.

SSL

문제점

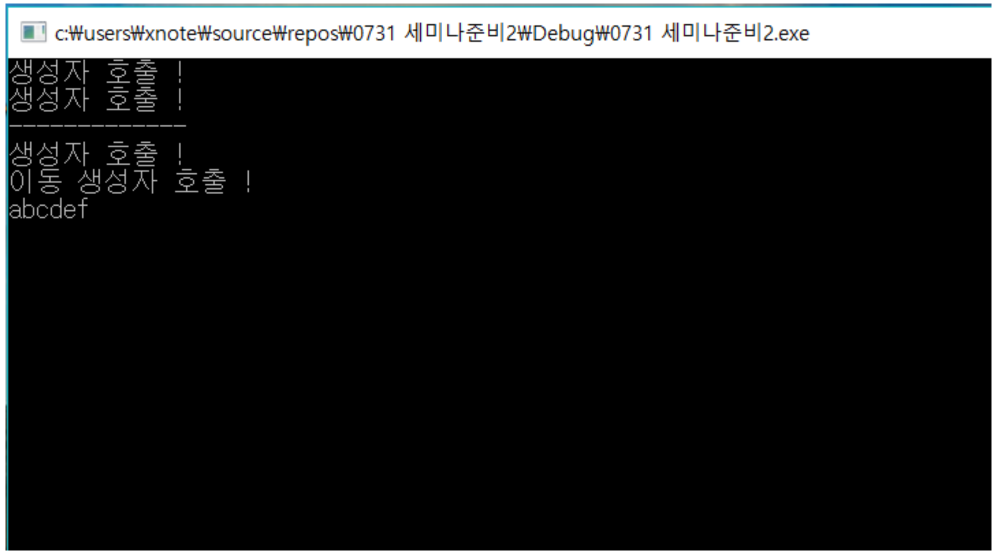
string_context 값을 변경 할 수 없음

⇒기존 복사 생성자를 사용 할 수없다.

해결 방법 : 우측값 레퍼런스(&&)를 통해 우측값 레퍼런스만 받게 한다.

SSL

```cpp
MyString::MyString(MyString&& str)
{
    cout << "이동 생성자 호출 !" << endl;

    string_length = str.string_length;
    string_content = str.string_content;
    memory_capacity = str.memory_capacity;

    str.string_content = nullptr;   // 임시 객체 소멸 시에 메모리를 해제하지 못하게 한다.

}


MyString::~MyString()
{
    if(string_content)
        delete[] string_content;
}
```

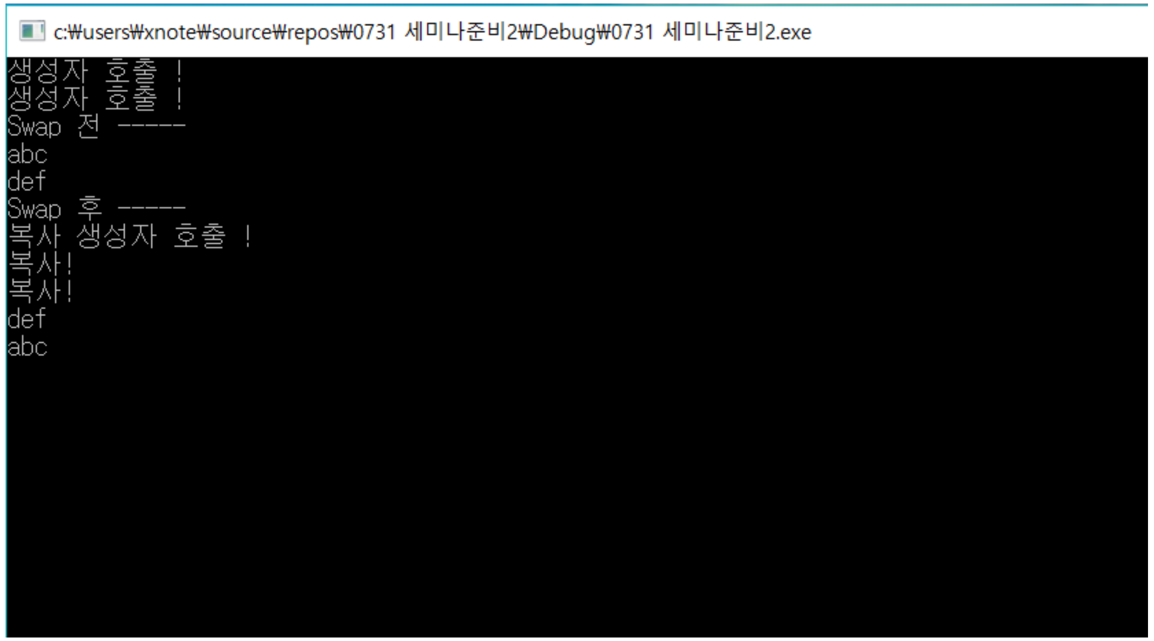

c:\users\xnote\source\repos\0731 세미나준비2\Debug\0731 세미나준비2.exe

```
생성자 호출 !
생성자 호출 !
-------------
생성자 호출 !
이동 생성자 호출 !
abcdef
```

SSL

4. Move 함수

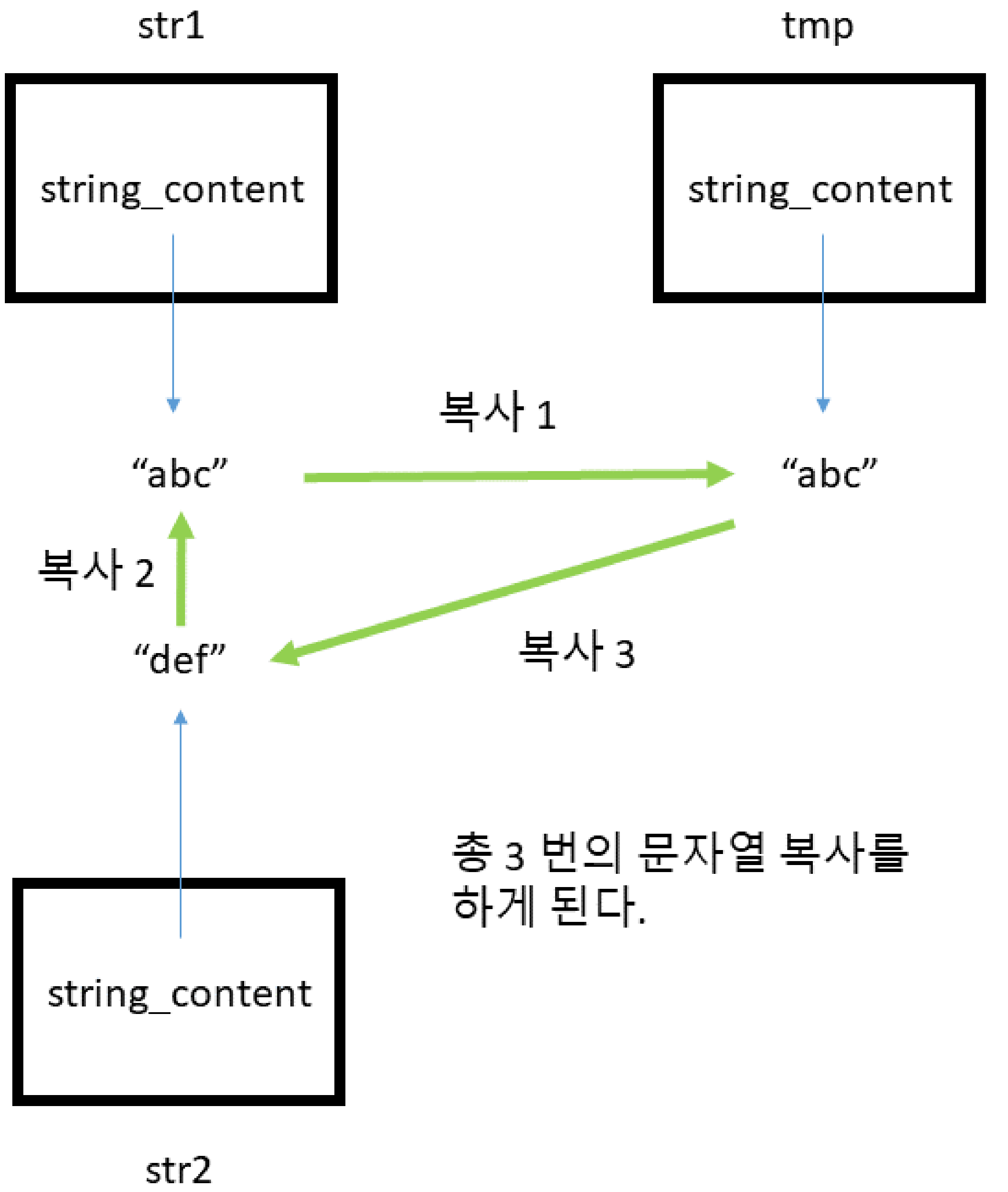
```cpp
template <typename T>
void my_swap(T &a, T &b)
{
    T tmp(a);
    a = b;
    b = tmp;

}

int main()
{

    MyString str1("abc");
    MyString str2("def");
    cout << "Swap 전 -----" << endl;
    str1.println();
    str2.println();

    cout << "Swap 후 -----" << endl;
    my_swap(str1, str2);
    str1.println();
    str2.println();
    _getch();

    return 0;
}
```



c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe

```
생성자 호출 !
생성자 호출 !
Swap 전 -----
abc
def
Swap 후 -----
복사 생성자 호출 !
복사!
복사!
def
abc
```

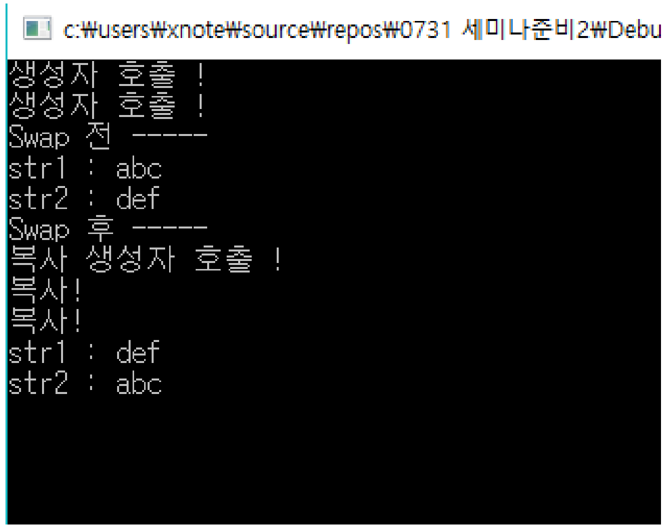
SSL



총 3 번의 문자열 복사를
하게 된다.

SSL

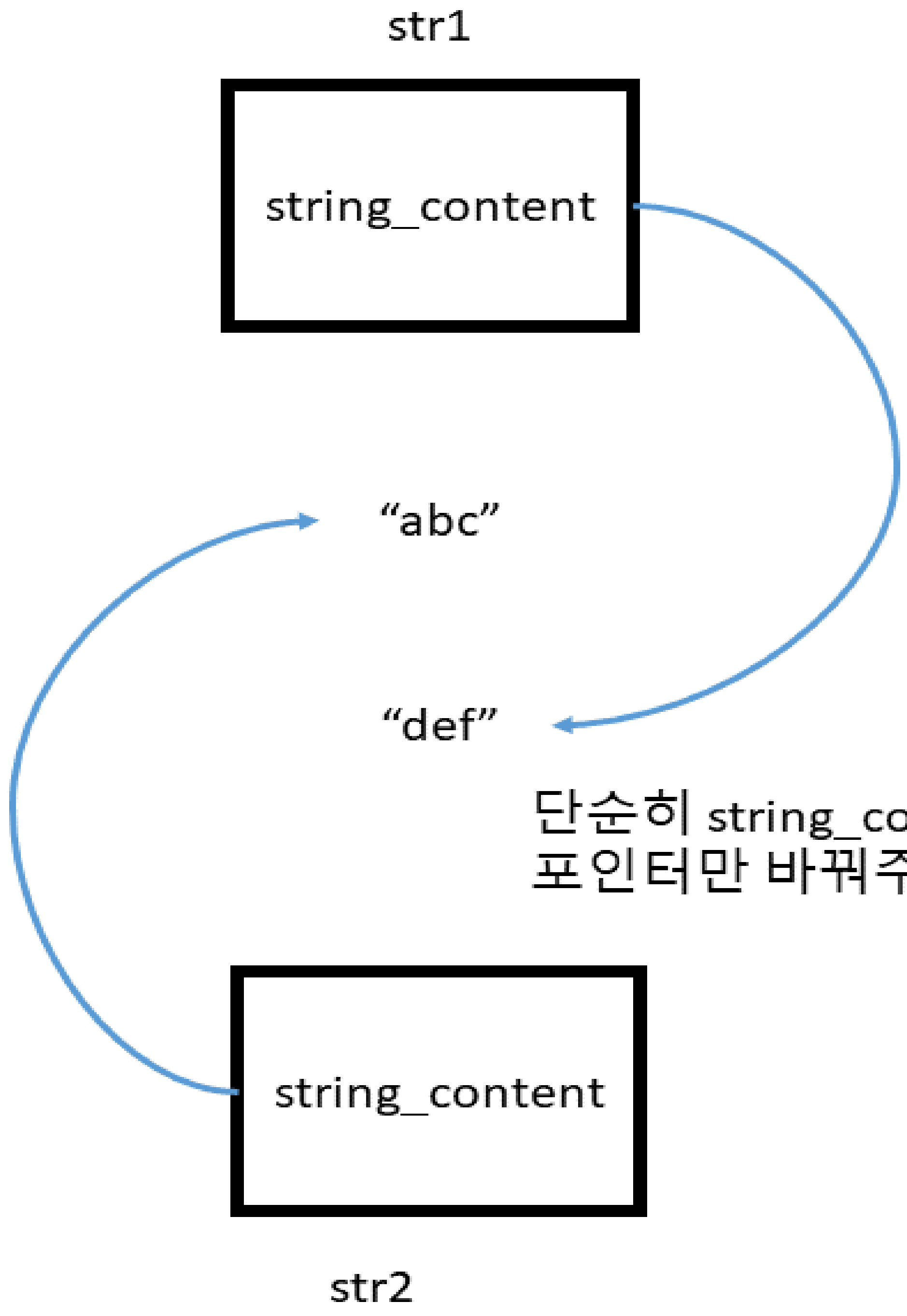
```cpp
template <typename T>
void my_swap(T &a, T &b)
{
    T tmp(a);
    a = b;
    b = tmp;
}

int main()
{
    MyString str1("abc");
    MyString str2("def");

    cout << "Swap 전 -----" << endl;
    cout << "str1 : "; str1.println();
    cout << "str2 : "; str2.println();

    cout << "Swap 후 -----" << endl;

    my_swap(str1, str2);

    cout << "str1 : "; str1.println();
    cout << "str2 : "; str2.println();

    _getch();
    return 0;
}
```



c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debu

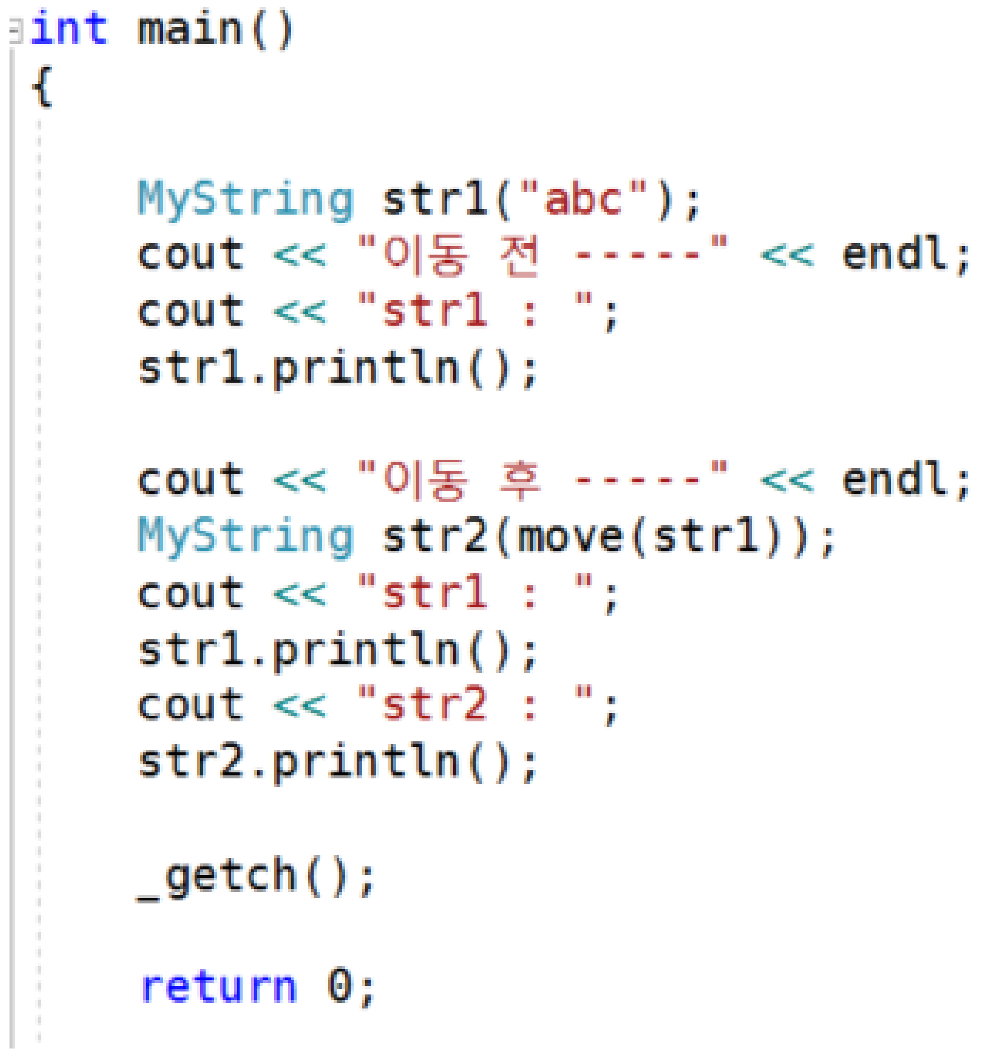
```
생성자 호출 !
생성자 호출 !
Swap 전 -----
str1 : abc
str2 : def
Swap 후 -----
복사 생성자 호출 !
복사!
복사!
str1 : def
str2 : abc
```

SSL

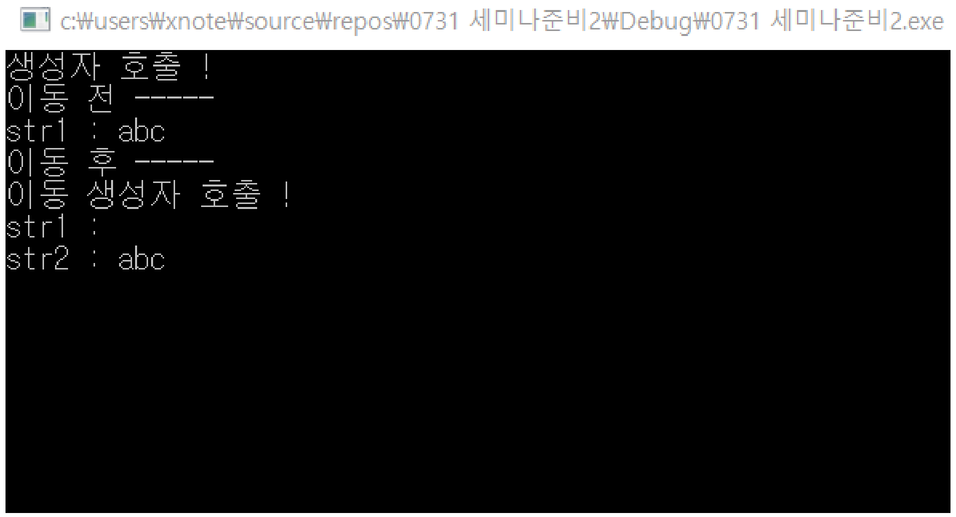


단순히 string_content
포인터만 바꿔주면 된다

SSL



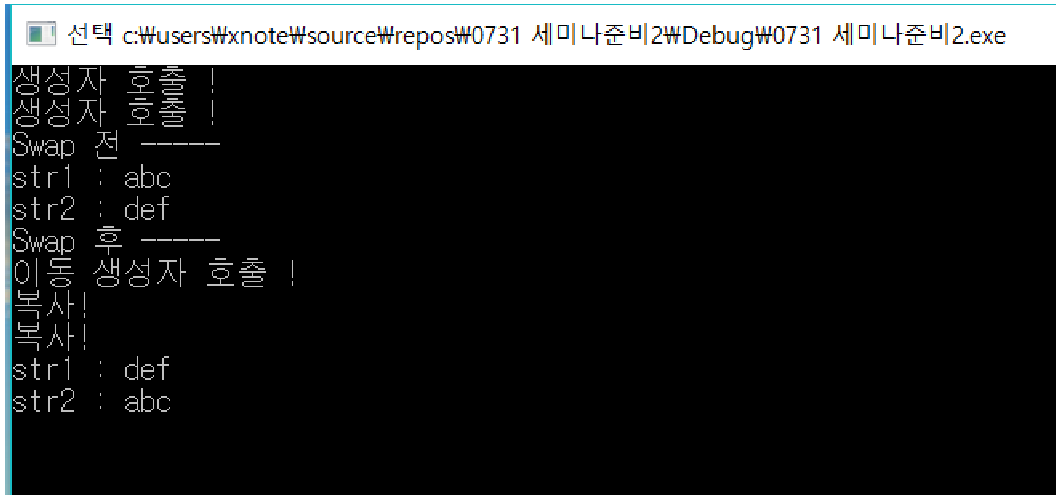
```cpp
int main()
{

    MyString str1("abc");
    cout << "이동 전 -----" << endl;
    cout << "str1 : ";
    str1.println();

    cout << "이동 후 -----" << endl;
    MyString str2(move(str1));
    cout << "str1 : ";
    str1.println();
    cout << "str2 : ";
    str2.println();

    _getch();

    return 0;
```



c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe

```
생성자 호출 !
이동 전 -----
str1 : abc
이동 후 -----
이동 생성자 호출 !
str1 :
str2 : abc
```

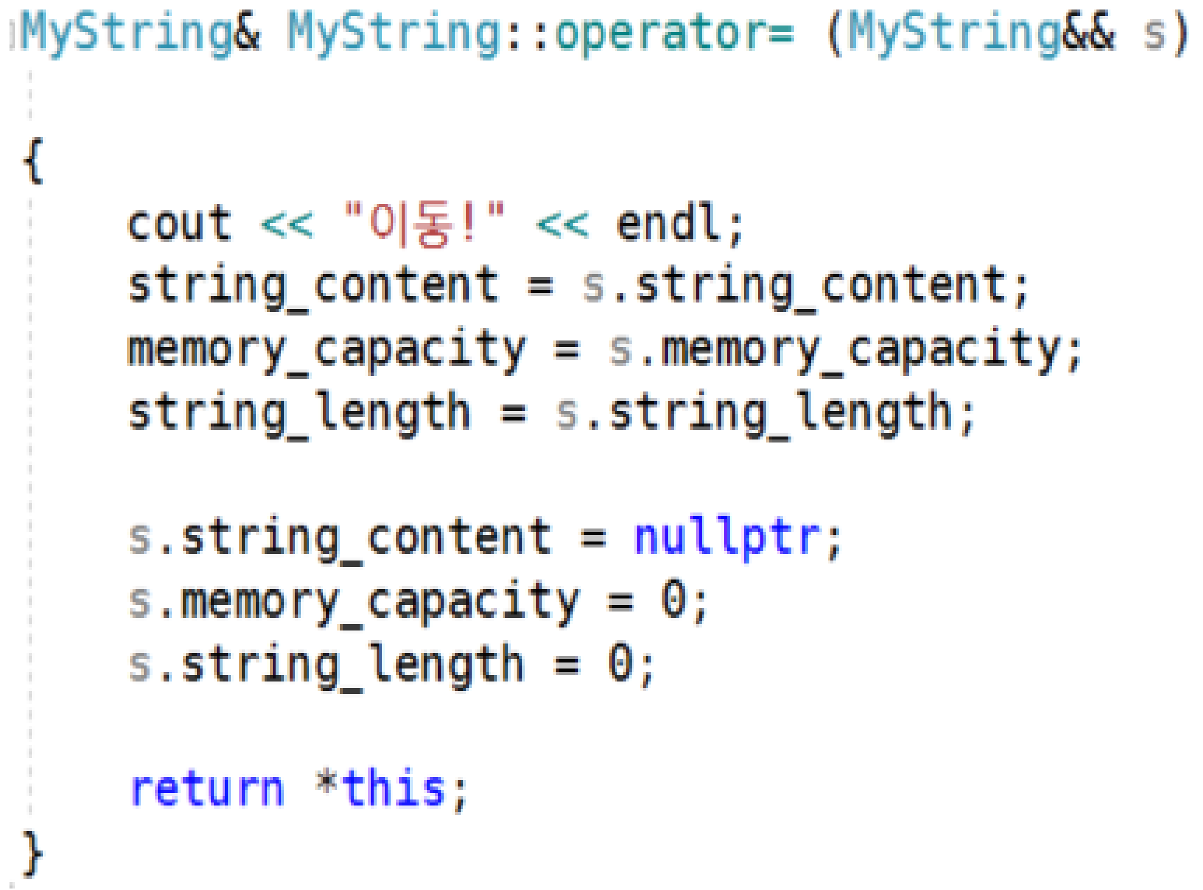
```
template <typename T>
void my_swap(T &a, T &b)
{
    T tmp(move(a));
    a = move(b);
    b = move(tmp);
}

int main()
{

    MyString str1("abc");
    MyString str2("def");

    cout << "Swap 전 -----" << endl;
    cout << "str1 : "; str1.println();
    cout << "str2 : "; str2.println();

    cout << "Swap 후 -----" << endl;

    my_swap(str1, str2);

    cout << "str1 : "; str1.println();
    cout << "str2 : "; str2.println();

    _getch();
    return 0;
}
```



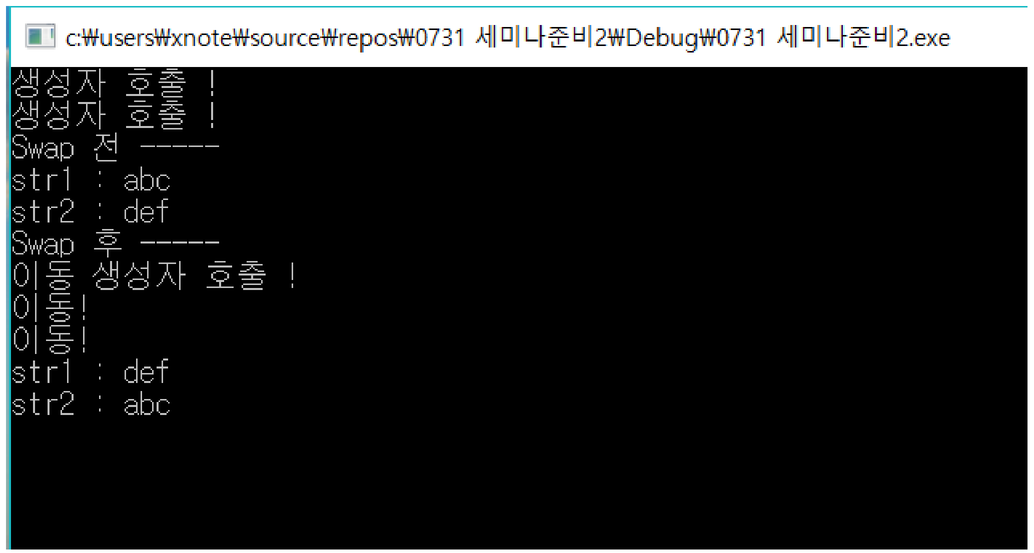
선택 c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe

```
생성자 호출 !
생성자 호출 !
Swap 전 -----
str1 : abc
str2 : def
Swap 후 -----
이동 생성자 호출 !
복사!
복사!
str1 : def
str2 : abc
```

SSL



```cpp
MyString& MyString::operator= (MyString&& s)
{
    cout << "이동!" << endl;
    string_content = s.string_content;
    memory_capacity = s.memory_capacity;
    string_length = s.string_length;

    s.string_content = nullptr;
    s.memory_capacity = 0;
    s.string_length = 0;

    return *this;
}
```



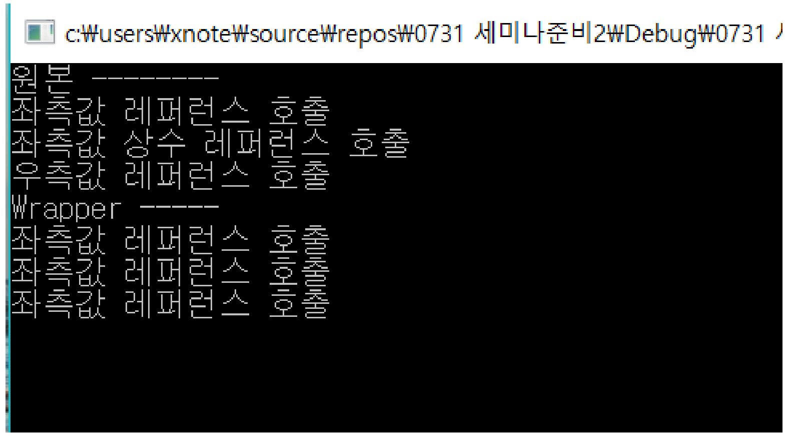
c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe

```
생성자 호출 !
생성자 호출 !
Swap 전 -----
str1 : abc
str2 : def
Swap 후 -----
이동 생성자 호출 !
이동!
이동!
str1 : def
str2 : abc
```

5. 완벽한 전달

```cpp
template <typename T>
void wrapper(T u)
{
    g(u);
}

class A {};

void g(A& a) {

    cout << "좌측값 레퍼런스 호출" << endl;
}

void g(const A& a) {

    cout << "좌측값 상수 레퍼런스 호출" << endl;
}

void g(A&& a) {

    cout << "우측값 레퍼런스 호출" << endl;

}
```

```cpp
int main()
{

    A a;
    const A ca;

    cout << "원본 --------" << endl;
    g(a);
    g(ca);
    g(A());

    cout << "Wrapper -----" << endl;
    wrapper(a);
    wrapper(ca);
    wrapper(A());

    _getch();
    return 0;
}
```

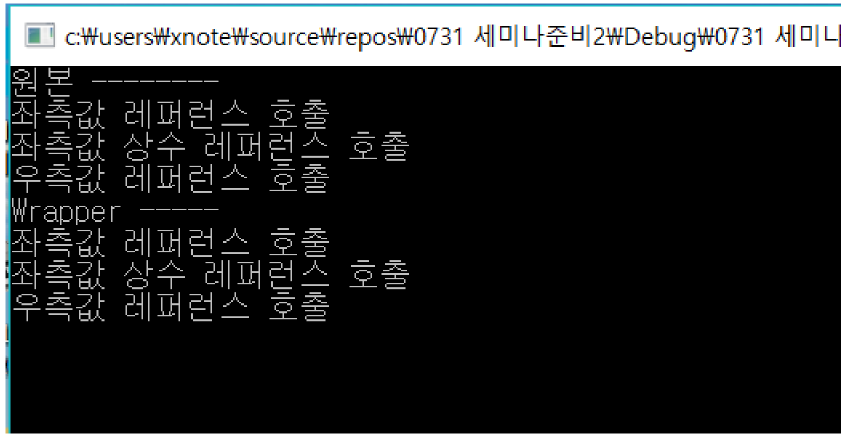


c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731

```
원본 --------
좌측값 레퍼런스 호출
좌측값 상수 레퍼런스 호출
우측값 레퍼런스 호출
Wrapper -----
좌측값 레퍼런스 호출
좌측값 레퍼런스 호출
좌측값 레퍼런스 호출
```

SSL

```cpp
template <typename T>
void wrapper(T&& u)
{
    g(forward<T>(u));
}

class A {};

void g(A& a) {

    cout << "좌측값 레퍼런스 호출" << endl;
}

void g(const A& a) {

    cout << "좌측값 상수 레퍼런스 호출" << endl;
}

void g(A&& a) {

    cout << "우측값 레퍼런스 호출" << endl;
}
```


c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나

```
원본 ---------
좌측값 레퍼런스 호출
좌측값 상수 레퍼런스 호출
우측값 레퍼런스 호출
Wrapper ------
좌측값 레퍼런스 호출
좌측값 상수 레퍼런스 호출
우측값 레퍼런스 호출
```

SSL

Forward 함수

typedef int& T;
T& r1; // int& &; r1 은 int&
T&& r2; // int & &&; r2 는 int&

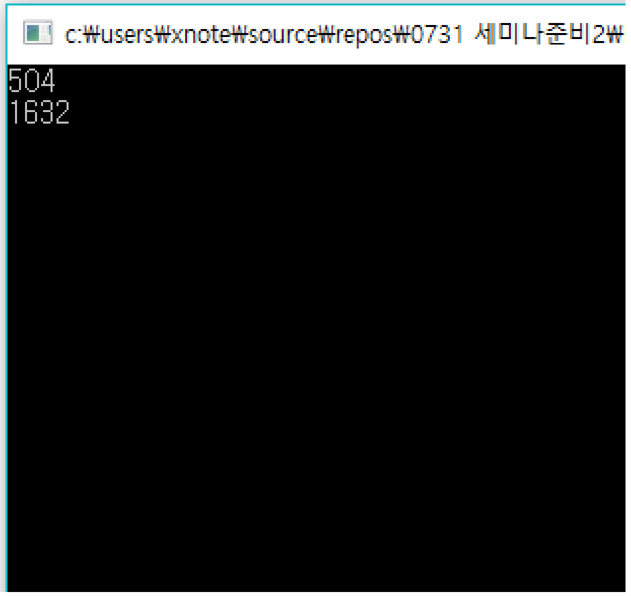
typedef int&& U
U& r3; // int && &; r3 는 int&
U&& r4; // int && &&; r4 는 int&&

=> 우측 레퍼런스 일때만 move 적용한 것 처럼 작동

SSL

6. 함수 객체

```
class Volume
{

public :
    double operator()(double x, double y, double z) { return x * y * z; }

    double operator()(Box& box)
    {
        return box.getLength() * box.getWidth() * box.getHeight();
    }
};

int main(void)
{
    Volume volume;

    double room{ volume(16, 12, 8.5) };

    cout << volume(7, 8, 9) << endl;
    cout << room << endl;

    _getch();
    return 0;
}
```



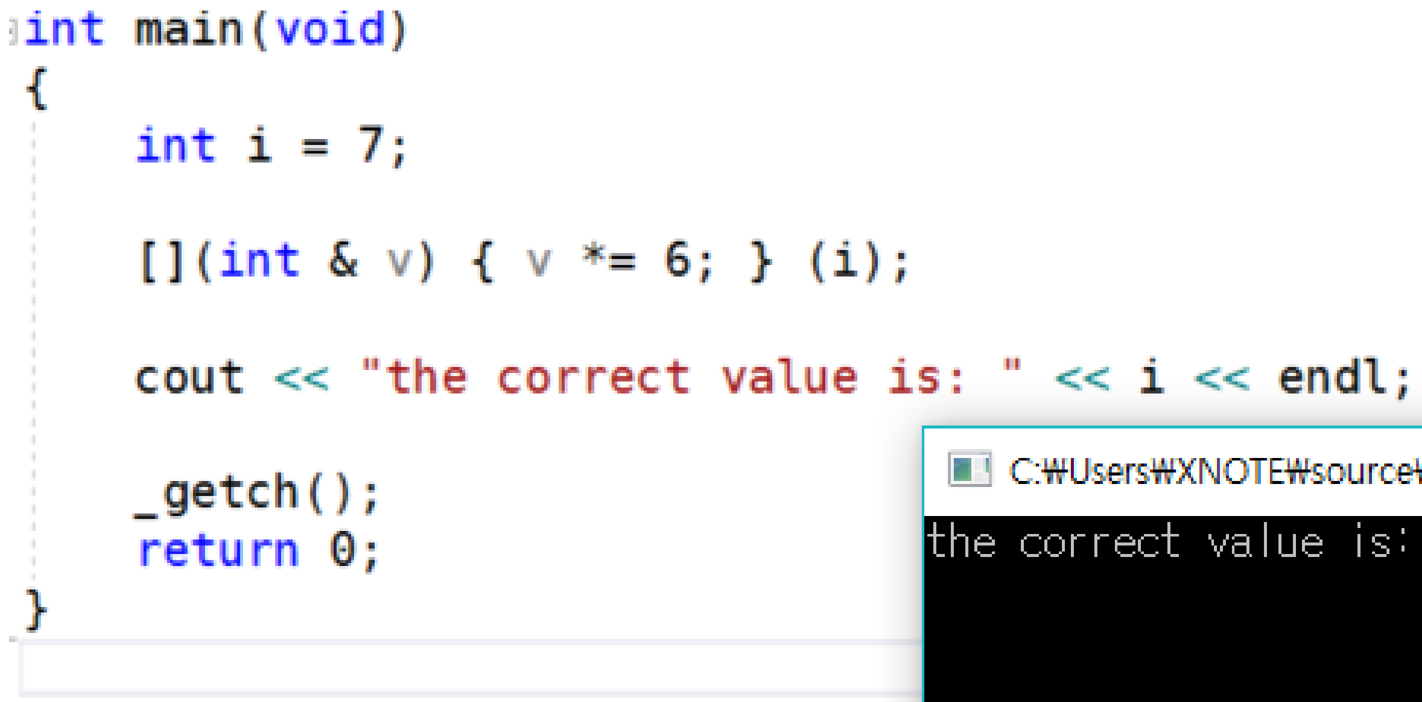
```
504
1632
```

SSL

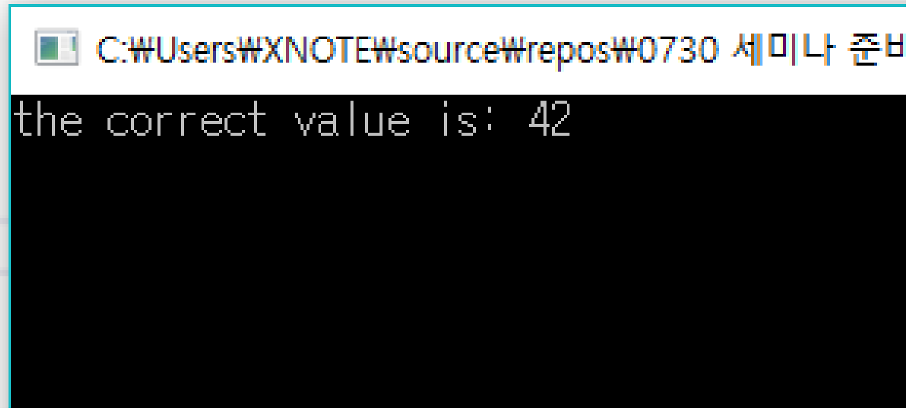
7. 람다 표현식

[캡쳐] (매개변수) ->반환형 {함수} (넘길 인자)

Ex)



```
int main(void)
{
    int i = 7;

    [](int & v) { v *= 6; } (i);

    cout << "the correct value is: " << i << endl;

    _getch();
    return 0;
}
```



C:₩Users₩XNOTE₩source₩repos₩0730 세미나 준비

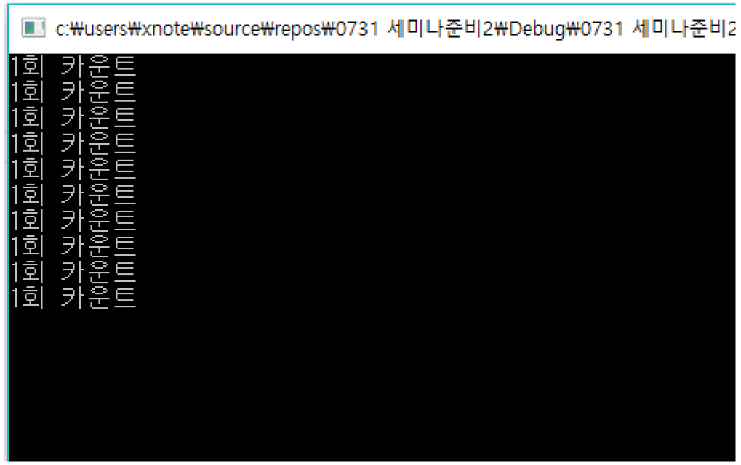
```
the correct value is: 42
```

SSL

캡쳐 : 람다 함수 내부에서 외부 변수를 캡쳐한다.

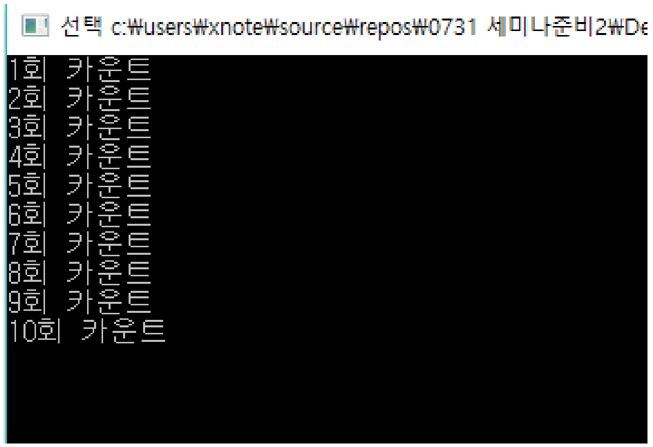
[=] : 모든 외부 변수를 값으로 전달받아서 캡쳐
[&] : 모든 외부 변수를 참조로 전달받아서 캡쳐

```cpp
class Lamda
{
public:
    int count;
};

int main()
{
    Lamda a;
    a.count = 1;

    for (int i = 0; i < 10; i++)
    {
        [=] {cout << a.count << "회 카운트" << endl; }();
    }
    _getch();
    return 0;
}
```



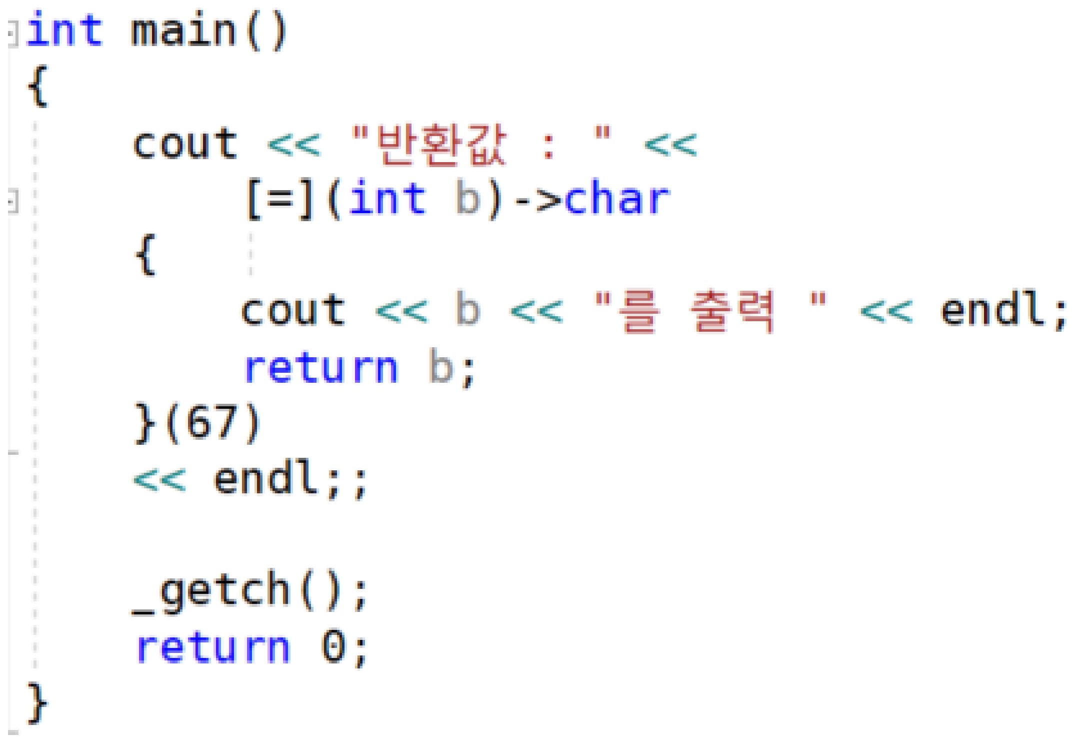
```
c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2
1회 카운트
1회 카운트
1회 카운트
1회 카운트
1회 카운트
1회 카운트
1회 카운트
1회 카운트
1회 카운트
1회 카운트
```

```cpp
int main()
{
    Lamda a;
    a.count = 1;

    for (int i = 0; i < 10; i++)
    {
        [&] {cout << a.count++ << "회 카운트" << endl; }();
    }
    _getch();
    return 0;
}
```

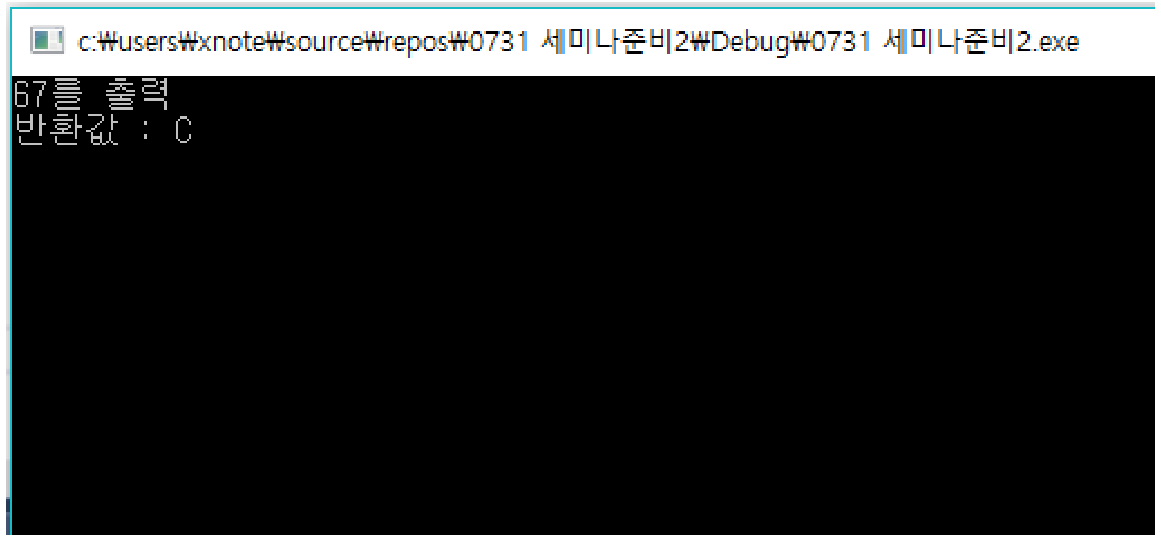


```
선택 c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩De
1회 카운트
2회 카운트
3회 카운트
4회 카운트
5회 카운트
6회 카운트
7회 카운트
8회 카운트
9회 카운트
10회 카운트
```

SSL



```
int main()
{
    cout << "반환값 : " <<
        [=](int b)->char
    {
        cout << b << "를 출력 " << endl;
        return b;
    }(67)
    << endl;;

    _getch();
    return 0;
}
```



c:₩users₩xnote₩source₩repos₩0731 세미나준비2₩Debug₩0731 세미나준비2.exe

```
67를 출력
반환값 : C
```

반환 값을 지정하지 않으면 return 값의 자료형으로 반환

SSL

참고문헌

1] 모던 C++ 입문
(저자 : 피터 고츠슐링)

2] http://itguru.tistory.com/

3] https://blog.naver.com/jidon333

4] https://msdn.microsoft.com/ko-kr/

SSL