

C++ Korea 제 6회 세미나

종합선물세트 제 2호



C++ Korea Gift Set

외계인이 만든

C++ Reference

Client & Server 개발자

김화수(金花秀)

flower_excel@naver.com

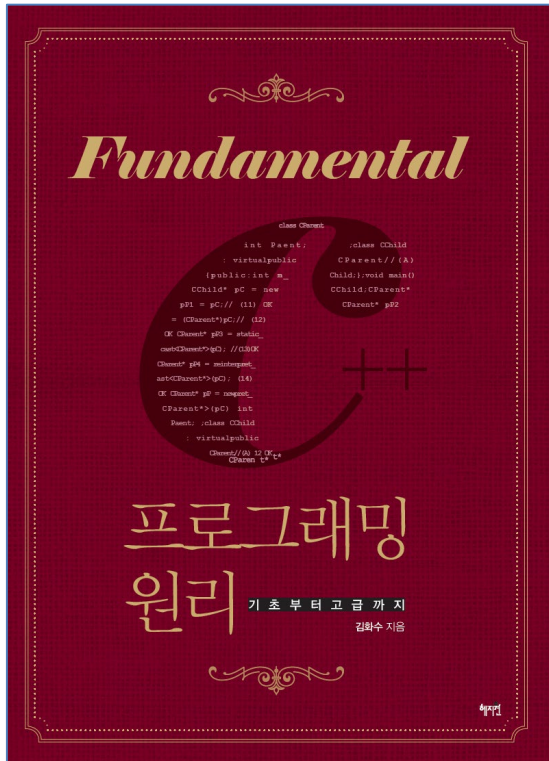


김화수(金花秀)

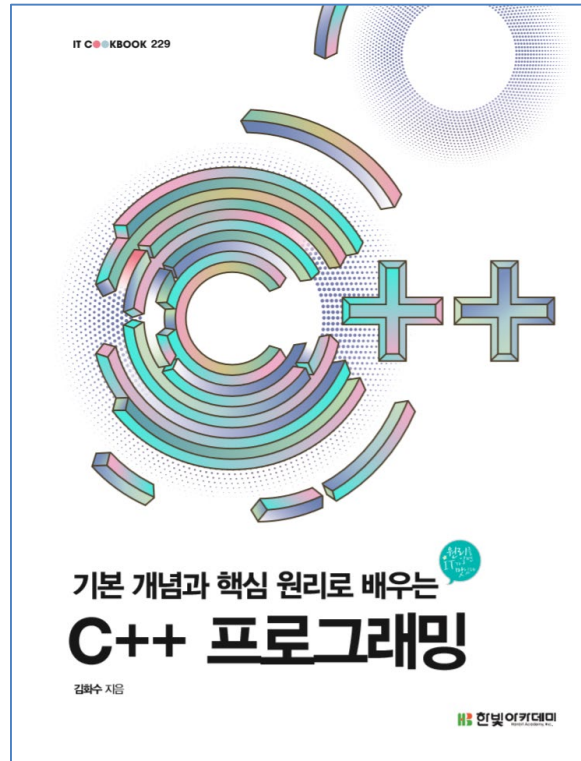
전) Naver NDrive 탐색기 개발

전) SK Telecom CLOUDBERRY 탐색기 개발

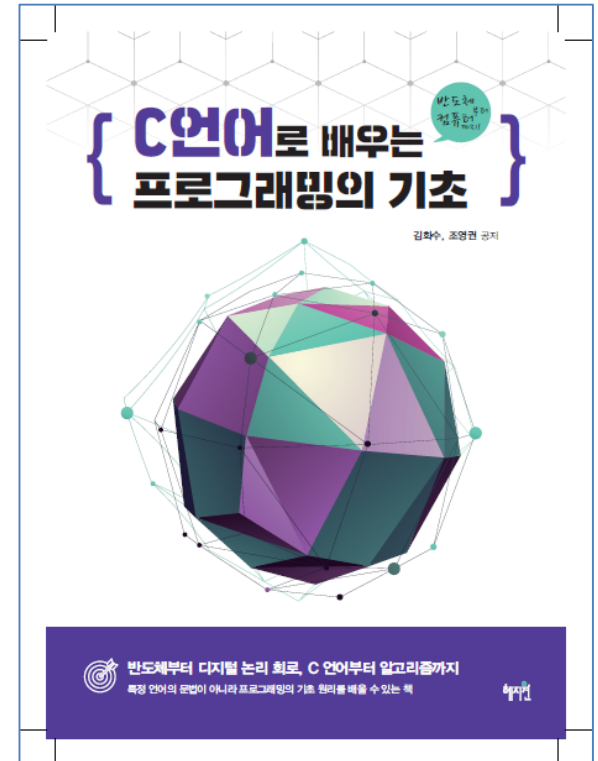
<저서>



(2015, 혜지원)



(2017, 한빛아카데미)



(2018, 혜지원)

알쏭달쏭한 Quiz

```
class CTest
{
public:
    CTest()
    { cout << "Default Constructor"; }

    CTest(const CTest& arg)
    { cout << "Copy Constructor"; }

    CTest(CTest&& arg)
    { cout << "Move Constructor"; }

    void operator = (const CTest& arg)
    { cout << "Copy Assignment"; }

    void operator = (CTest&& arg)
    { cout << "Move Assignment"; }
};
```

```
CTest GetTest()
{
    static CTest s_Test;
    return s_Test;
}

int main()
{
    CTest t = GetTest(); // ?
    return 0;
}
```

- 1) Default Constructor
- 2) Copy Constructor
- 3) Move Constructor
- 4) Copy Assignment
- 5) Move Assignment

일반적인 반응들...

- 잘못 들어왔다. 옆방 갈 걸...
- &&가 뭐지? 오타인가?
- Move가 뭐지? 최신 C++인가보다.
- 너무 쉽군! Move Constructor
- **저런 거 안다고 도움이 될까?**

이번 세미나의 핵심

- C++ Reference의 발전사를 살펴보자!
- C++ Reference의 의도를 살펴보자!
- RValue Reference를 알아보자!
- Compiler 최적화를 살펴보자!
- C++ Code를 C++답게 구사하기!

인자 전달(Value vs Pointer)

```
class CTest
```

```
{
```

```
public:
```

```
    int m_Array[4];
```

```
};
```

```
void FValue(CTest arg) { }
```

```
void FPointer(CTest* p) { }
```

```
int Test()
```

```
{
```

```
    CTest t;
```

```
    FValue(t);
```

```
    FPointer(&t);
```

```
}
```

```
lea    rax,[rsp+30h]
```

```
lea    rcx,[t]
```

```
mov     rdi,rax
```

```
mov     rsi,rcx
```

```
mov     ecx,10h
```

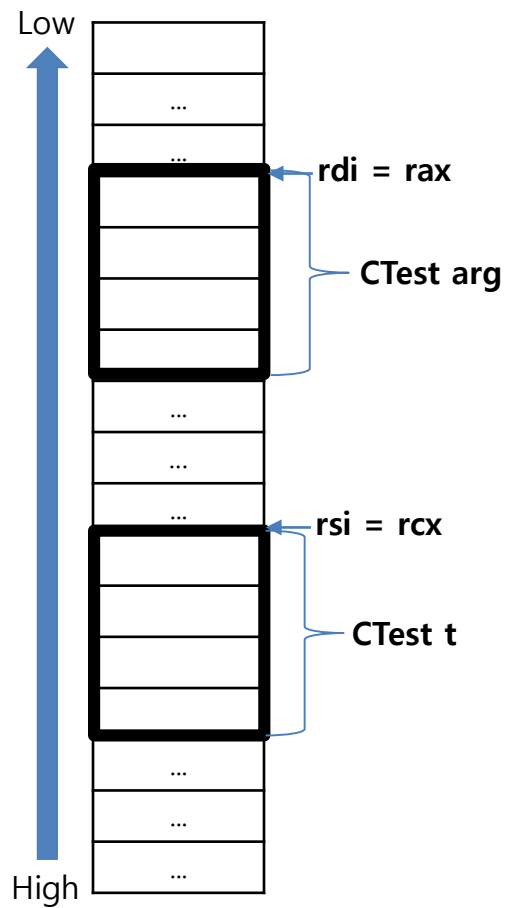
```
rep movs byte ptr [rdi],byte ptr [rsi]
```

```
lea     rcx,[rsp+30h]
```

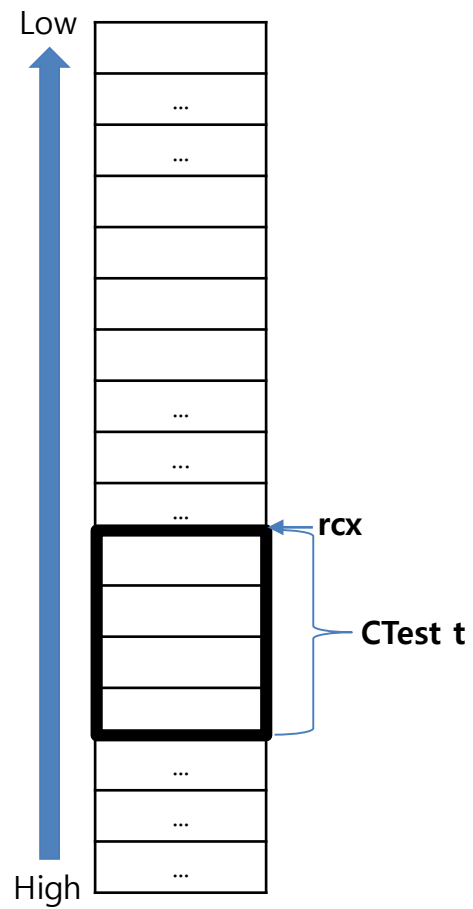
```
call    FValue
```

```
lea     rcx,[t]
```

```
call    FPointer
```



(a) 값 인자 전달



(b) 포인터 인자 전달

Const Pointer Parameter

```
class CTest
{
public:
    int m_Array[4];
};

void FPointer(CTest* p) { }
void FConstPointer(const CTest* p)
{ }

int Test1()
{
    const CTest ct;
    FPointer(&ct);
    FConstPointer(&ct);

    CTest t;
    FConstPointer(&t);
}
```

```
CTest GetTest()
{
    CTest t;
    return t;
}

int Test2()
{
    FuncConstPointer(&CTest());
    FuncConstPointer(&GetTest());
}
```

G++ Error
VC++ OK

Reference Parameter (특히 const T&)

```
class CTest
{
public:
    int m_Array[4];
};

void FRef(CTest& arg) { }
void FConstRef(const CTest& arg) { }

int Test()
{
    CTest t;
    FRef(t);

    const CTest ct;
    FRef(ct);
    FConstRef(ct);

    FRef(CTest());
    FConstRef(CTest());
}
```

```
void FConstRef(const int& arg) { }

int Test()
{
    int i = 1;
    FConstRef(i);


    const int ci = 1;
    FConstRef(ci);

    FConstRef(1); // 상수 Literal

    return 0;
}
```



```
lea rcx, [t]
call FRef
```



```
lea rcx, [rsp+80h]
call CTest::CTest
mov rcx, rax
call FConstPointer
```

Reference Parameter 함수 중복 정의

```
class CTest
{
public:
    void noop() { }
    int m_Array[4];
};
```

```
void Func(CTest& arg)
{
    arg.m_Array[0] = 3;
    arg.noop();
}
```

```
void Func(const CTest& arg)
{
    arg.m_Array[0] = 3;
    arg.noop();
}
```

```
int Test()
```

```
{
    CTest t;
    Func(t);
}
```

```
const CTest ct;
Func(ct);
```

```
Func(CTest());
}
```

CTest().noop(); // 기존 방식

```
void InvokeNoop(CTest& arg)
{
    arg.noop();
}
```

InvokeNoop(CTest()); ?

대학생들이 미분방정식 숙제를 하는 방법

1. 열심히 한다.
2. 열심히 숙제를 한 친구의 숙제를 베낀다.
3. 숙제를 이미 베낀 친구의 숙제를 베낀다.
4. 남의 숙제를 얻어서 내 이름으로 바꾼다.

직접 숙제하기, 베끼기

```
class CReport
{
public:
    CReport(const char* Name,
            const char* Text = NULL)
    {
        m_Name = Name;
        m_Text = Text;
    }

    ~CReport()
    {
        if(m_Text)
        {
            delete [] m_Text;
            m_Text = NULL;
        }
    }

    const char* m_Name;
    const char* m_Text;
};
```

```
const char* DoReport()
{
    char* pText = new char[1024];
    pText[0] = 'W';
    strcat(pText, "x + 1 = 3, x = 2");
    return pText;
}

void Copy()
{
    CReport* pReport = new
    CReport("KHS", DoReport());

    CReport copy("Copycat");
    copy.m_Text = new char[1024];
    strcpy((char*)copy.m_Text,
           pReport->m_Text);

    delete pReport;
}
```

남의 것에 내 이름 쓰기

```
void DangerMove1()
{
    CReport* pReport = new
    CReport("KHS", DoReport());

    CReport move("Move",
                pReport->m_Text);

    delete pReport;
}
```

```
void DangerMove2()
{
    CReport* pReport = new
    CReport("KHS", DoReport());

    CReport move("Move",
                pReport->m_Text);

    pReport->m_Text = NULL;

    // 여기서부터 pReport는 무효화

    delete pReport;
}
```

pReport가 더 이상 사용되지 않는다면
괜찮지 않을까?
가령 학교를 자퇴한 친구의 숙제를
내 이름으로 바꾼다면...

RValue Reference

```
class CTest
{
public:
    void noop() {}
    int m_Array[4];
};
```

```
void Func(CTest& arg) { }
```

```
void Func(const CTest& arg) { }
```

```
void Func(CTest&& arg) { }
```

```
void InvokeNoop(CTest&& arg)
{
    arg.noop();
}
```

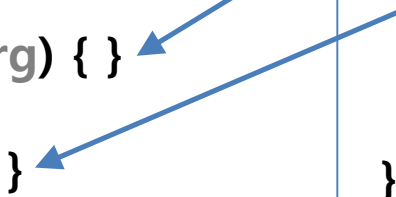
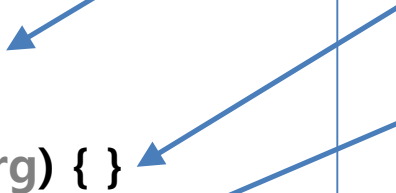
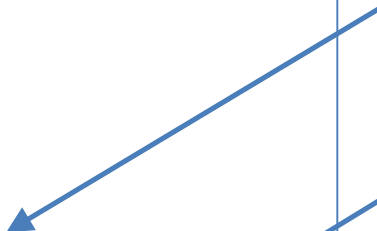
```
void Test()
{
```

```
    CTest t;
    Func(t);
```

```
    const CTest ct;
    Func(ct);
```

```
    Func(CTest());
```

```
    InvokeNoop(CTest());
}
```



제대로 숙제를 해볼까요? 1

```
class CReport
{
public:
    CReport(const char* Name,
            const char* Text = NULL)
    {
        m_Name = Name;
        m_Text = Text;
    }

    CReport(const CReport& arg)
    {
        m_Text = new char[1024];
        strcpy((char*)m_Text,
               arg.m_Text);
    }

    CReport(CReport&& arg)
    {
        m_Text = arg.m_Text;
        arg.m_Text = NULL;
    }
}
```

```
void operator = (const CReport& arg)
{
    m_Text = new char[1024];
    strcpy((char*)m_Text, arg.m_Text);
}

void operator = (CReport&& arg)
{
    m_Text = arg.m_Text;
    arg.m_Text = NULL;
}

~CReport()
{
    if(m_Text)
    {
        delete [] m_Text;
        m_Text = NULL;
    }
}

const char* m_Name;
const char* m_Text;
};
```

제대로 숙제를 해볼까요? 2

```
void MoveReport()
{
    CReport HW("KHS", DoReport());

    CReport copy1(HW);
    copy1.m_Name = "Copycat1";

    CReport move1 =
    CReport("KHS", DoReport());
    move1.m_Name = "Move1";

    CReport copy2("Copycat2");
    copy2 = HW;

    CReport move2("Move2");
    move2 = CReport("KHS",
                    DoReport());
}
```

The diagram illustrates the mapping between code snippets in the `MoveReport()` function and the `CReport` class methods. A vertical line separates the two columns, and arrows point from the code to the corresponding method signature.

- `CReport HW("KHS", DoReport());` maps to `CReport(const char* Name, const char* Text = NULL);`
- `CReport copy1(HW);` maps to `CReport(const CReport& arg);`
- `CReport move1 = CReport("KHS", DoReport());` maps to `CReport(CReport&& arg);`
- `copy2 = HW;` maps to `void operator = (const CReport& arg);`
- `move2 = CReport("KHS", DoReport());` maps to `void operator = (CReport&& arg);`

std::move

```
class CTest
{
public:
    void noop() {}
    int m_Array[4];
};

void g(CTest& arg) { }

void g(CTest&& arg) { }

void Func(CTest&& arg)
{
    CTest* pTest = &arg;
    g(arg);
    g(std::move(arg));
}
```

```
void Test()
{
    CTest t;
    Func(t);
    Func((CTest&&)t);
    Func(std::move(t));

    const CTest ct;
    Func(ct);
    Func((CTest&&)ct);
    Func(std::move((CTest&)ct));

    Func(CTest());
    Func(std::move(CTest()));
}
```

std::forward

```
void g(CTest& arg) { }  
void g(CTest&& arg) { }
```

```
void Test()  
{  
    CTest t;  
    g(std::forward<CTest>(t));  
    g(std::forward<CTest&>(t));  
    g(std::forward<CTest&&>(t));  
  
    g(std::forward<CTest>(CTest()));  
    g(std::forward<CTest&>(CTest()));  
    g(std::forward<CTest&&>(CTest()));  
  
    g((CTest&)CTest());  
}
```

```
// forward an lvalue as  
//     either an lvalue or an rvalue  
template <class _Ty>  
_Ty&& forward(remove_reference_t<_Ty>& _Arg)  
{  
    return static_cast<_Ty&&>(_Arg);  
}  
  
// forward an rvalue as an rvalue  
template <class _Ty>  
_Ty&& forward(remove_reference_t<_Ty>&& _Arg)  
{  
    static_assert(!is_lvalue_reference_v<_Ty>,  
                  "bad forward call");  
    return static_cast<_Ty&&>(_Arg);  
}
```

(요약) RValue만 LValue로 변환 불가
주로 Template의 Universal Reference에서 사용

컴파일러 최적화 1

```
class CTest
{
public:
    CTest()
    {
        cout << "Default Constructor";
    }

    CTest(const CTest& arg)
    {
        cout << "Copy Constructor";
    }

    CTest(CTest&& arg)
    {
        cout << "Move Constructor";
    }

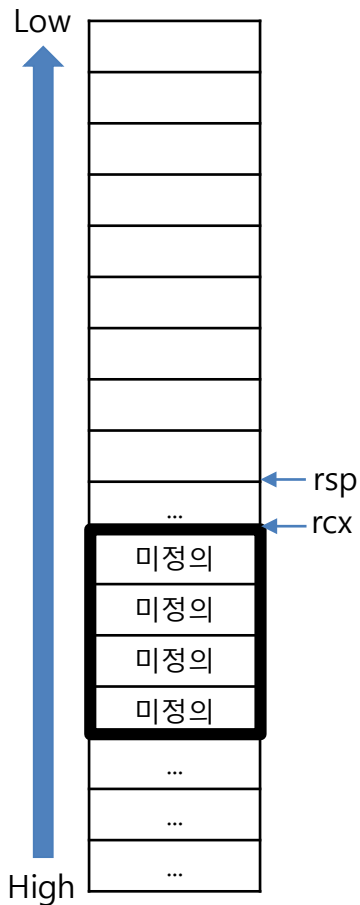
    int m_Val;
    char m_Array[12];
};
```

```
CTest GetTest(); // 값 반환 함수
void Func(CTest arg) { }

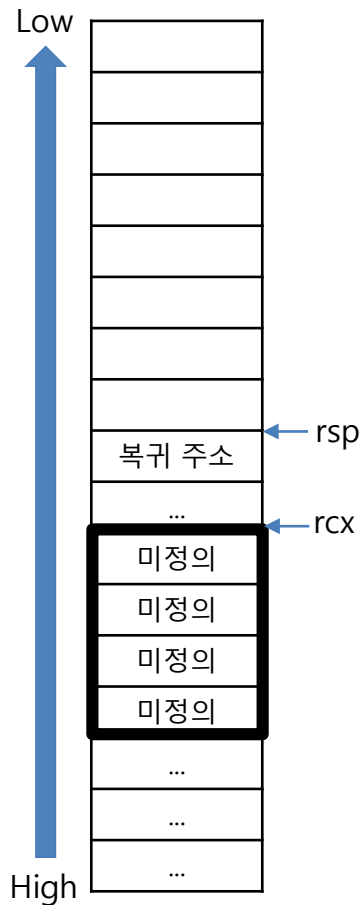
void Test()
{
    GetTest();
    lea rcx,[rsp+38h] // rcx = RA
    call GetTest

    CTest t = GetTest();
    lea rcx,[t] // t = rcx = RA
    call GetTest

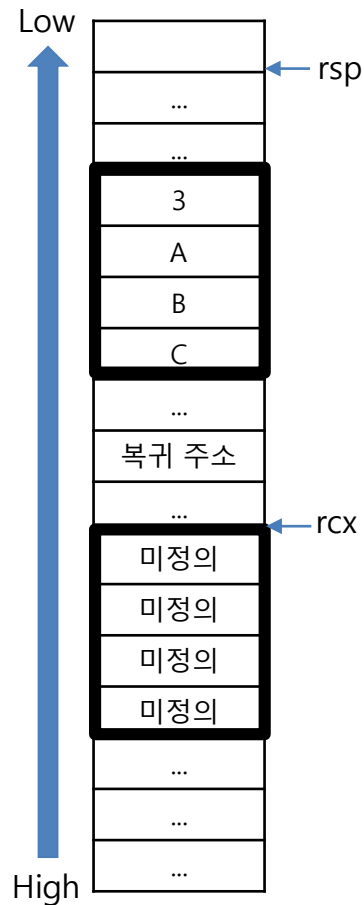
    Func(GetTest());
    lea rax,[rsp+28h]
    mov qword ptr [rsp+20h],rax
    mov rcx,qword ptr [rsp+20h]
    call GetTest
    mov rcx,rax // arg = rcx = rax = RA
    call Func
}
```



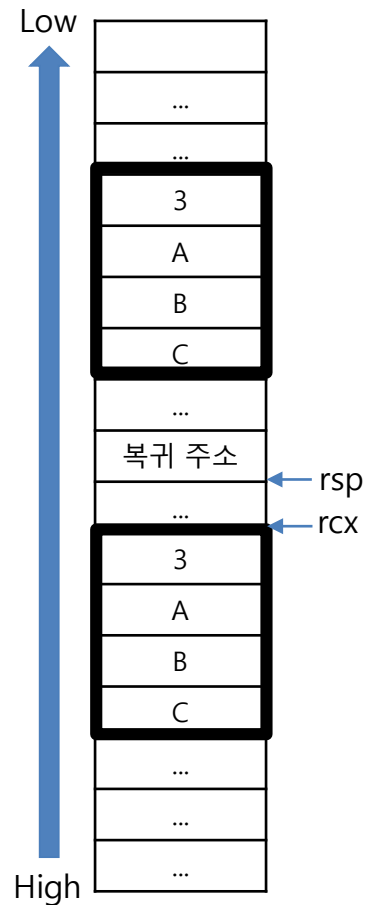
(a) 함수 호출 전



(b) 함수 호출 직후



(c) 함수 스택 사용



(d) 함수 반환

컴파일러 최적화 2

```
CTest GetTest1( )
```

```
{  
    CTest t;  
    return t;  
}
```

```
CTest GetTest2( )
```

```
{  
    CTest t;  
    if(1)    // NRVO 불가  
    {  
        return t;  
    }  
    return t;  
}
```

```
CTest GetTest3( )
```

```
{  
    return CTest(...);  
}
```

```
CTest GetTest4( )
```

```
{  
    static CTest s_Test;  
    return s_Test;  
}
```

1) NRVO

rcx 기준으로 CTest t 생성
CTest() 호출

2-a) Move Constructor 정의

return t; -> return std::move(t);
CTest(CTest&& arg) 호출,

2-b) Move Constructor 미정의

CTest(const CTest& arg) 호출

3) RVO

rcx 기준으로 CTest(...) 호출

4) Copy

CTest(const CTest& arg) 호출

Q & A

감사합니다.