

클래스, 함수, 템플릿

# C++ 2주차

# 목차

- 클래스
  - 클래스와 구조체의 차이
  - 클래스의 메모리 구조
  - 생성자와 소멸자
  - 초기화 리스트
  - 상속
- 함수
  - 함수 호출 규약
  - 네임 데코레이션
  - 클래스 멤버 함수
  - 가상 함수
- 템플릿
  - 템플릿

# 클래스 : 클래스와 구조체의 차이

- C의 구조체 : 타입이 서로 다른 데이터들을 한 곳에 모아놓은 확장 자료형
  - C의 구조체는 함수를 멤버로 가질 수 없음
  - C 설계 팁 : 함수 포인터는 포인터 이므로 이것을 이용해 클래스와 비슷하게 설계 할 수 있음
- C++의 클래스 : C의 구조체에 멤버 함수를 추가한 것
- C++의 클래스와 구조체의 차이
  - 기본 접근 지정자가 private(클래스)이냐 public(구조체)이냐를 제외하고 완전히 동일함  
->C++의 구조체는 멤버 함수를 가질 수 있음(그렇다고 이렇게 사용하는거 권장 안함)

# 클래스 : 클래스의 메모리 구조

- 메모리 배치 규칙
  - 선언 순서대로 올라감
  - static 변수는 전역변수가 저장되는 영역(BSS)에
  - 함수는 Text(Code)영역에

- 클래스도 C구조체 처럼 바이트패딩 일어남
  - 컴파일러 옵션(/Zp)를 통해 바꿀 수 있음

```
int main()
{
    TestClass t;
    t.mChar = 'A';
    t.mInt = 1;
    t.mDouble = 0.5;
}

00001713 call     @__UncheckForDebuggerJustMyC
          TestClass t;
          t.mChar = 'A';
00001718 mov     byte ptr [t],41h
          t.mInt = 1;
0000171C mov     dword ptr [ebp-10h],1
          t.mDouble = 0.5;
00001723 movsd   xmm0,mmword ptr [__real@3f
0000172B movsd   mmword ptr [ebp-0Ch],xmm0
          }
00001730 xor     eax,eax
```

```
class TestClass
{
public:
    char mChar;
    int mInt;
    double mDouble;
    static int mSInt;
    int MemberFunc()
    {
        return mInt;
    }
};
```

- 클래스에 접근할 때 오프셋 정보 사용(BP로 부터의 상대적 위치)
- 클래스의 크기는 0보다 커야함
- 만약 가상함수를 멤버로 가질경우 \_vfptr 이라는 가상함수 테이블을 가르키는 포인터가 추가됨

mChar
mInt
mDouble

# 클래스 : 생성자와 소멸자

- 객체가 생성될 때 생성자가 호출되고 소멸될 때 소멸자가 호출됨
  - > 초기화 작업이나 자원 마무리를 처리하는데 용이함
- 암시적 생성자와 소멸자
  - 생성자나 소멸자가 명시적으로 정의되어 있지 않고 생성자나 소멸자가 필요 없다면
    - > 컴파일러는 생성자나 소멸자를 암시적으로 생성하지 않음
  - 생성자나 소멸자가 명시적으로 정의되어 있지 않고 생성자나 소멸자가 필요 하다면
    - > 컴파일러는 생성자나 소멸자를 암시적으로 생성함
- 암시적으로 생성자와 소멸자가 생성되는 경우
  - 상위 클래스의 생성자와 소멸자가 정의되어 있는 경우
    - > 상위클래스 생성자부터 실행되는데 왜?
  - 복사 생성자나 복사대입 생성자가 필요한 경우

# 클래스 : 생성자와 소멸자

- 생성자와 소멸자의 호출 순서

- 결과

- 상위클래스 생성자
- 하위클래스 생성자
- 하위클래스 소멸자
- 상위클래스 소멸자

- 실제 동작

- 상위클래스 생성자 호출 전 하위클래스 생성자 호출, 블록부분 실행 전 선처리 영역이 존재해 거기서 상위클래스 및 멤버가 클래스인 경우 멤버의 생성자 호출
- 하위클래스 소멸자 호출 후 블록이 끝나면 후처리 영역이 존재해 상위클래스 소멸자 호출

```
#include <iostream>

using namespace std;

class Base
{
public:
    Base() { cout << "Base - Constructor" << endl; };
    ~Base() { cout << "Base - Destructor" << endl; };
};

class Derived : public Base
{
public:
    Derived() { cout << "Derived - Constructor" << endl; };
    ~Derived() { cout << "Derived - Destructor" << endl; };
};

int main()
{
    Derived *dc = new Derived();
    delete dc;
}
```

```
C:\Users\Hyunjin Park\so
Base - Constructor
Derived - Constructor
Derived - Destructor
Base - Destructor
```

# 클래스 : 생성자와 소멸자

- virtual 소멸자
  - RTTI(Run-Time Type Information)
  - OOP의 다형성
- 상속 구조가 아니라면?
  - > 오버헤드만 증가됨

```
#include <iostream>

using namespace std;

class Base
{
public:
    Base() { cout << "Base - Constructor" << endl; };
    ~Base() { cout << "Base - Destructor" << endl; };
};

class Derived : public Base
{
public:
    Derived() { cout << "Derived - Constructor" << endl; };
    ~Derived() { cout << "Derived - Destructor" << endl; };
};

int main()
{
    Base *dc = new Derived();
    delete dc;
}
```

C:\Users\Hyunjin Park\source\repos\test\Debug\test.exe

Base - Constructor  
Derived - Constructor  
Base - Destructor

```
#include <iostream>

using namespace std;

class Base
{
public:
    Base() { cout << "Base - Constructor" << endl; };
    virtual ~Base() { cout << "Base - Destructor" << endl; };
};

class Derived : public Base
{
public:
    Derived() { cout << "Derived - Constructor" << endl; };
    virtual ~Derived() { cout << "Derived - Destructor" << endl; };
};

int main()
{
    Base *dc = new Derived();
    delete dc;
}
```

C:\Users\Hyunjin Park\source\repos\test\Debug\test.exe

Base - Constructor  
Derived - Constructor  
Derived - Destructor  
Base - Destructor

# 클래스 : 생성자와 소멸자

- 복사 생성자와 복사 대입 연산자

```
#include <iostream>

using namespace std;

class TestClass
{
public:
    TestClass() {} //기본 생성자
    TestClass(const TestClass& Obj) {} //복사 생성자
    TestClass& operator = (const TestClass& Obj) //복사 대입 연산자
    { return *this; }
};

int main()
{
    TestClass t0;
    TestClass t1(t0); //복사 생성자 호출
    TestClass t2 = t0; //복사 생성자 호출
    TestClass t3; t3 = t0; //복사 대입 연산자 호출
}
```



# 클래스 : 생성자와 소멸자

- 복사 생성자 : 객체의 값에 의한 호출시에도 복사 생성자 호출됨 -> 성능하락
- 암시적 복사 생성자
  - 부모 클래스의 복사 생성자 호출
  - 멤버가 클래스타입일 경우 복사 생성자 호출
  - 멤버가 기본 타입일 경우 메모리 복사
  - 멤버가 배열일 경우 원소의 타입에 따라 복사 생성자 호출 또는 메모리 복사 수행
  - 멤버가 참조 타입일 경우 대상 복사
- 명시적 복사 생성자
  - 부모클래스 기본 생성자 호출
  - 멤버가 클래스타입일 경우 기본 생성자 호출
  - 이후 프로그래머가 정의한 동작 실행

# 클래스 : 생성자와 소멸자

- 복사 대입 연산자
  - > 연산자는 일반 함수이므로 생성자처럼 선처리나 후처리 영역 존재하지 않음
  - 암시적 복사 대입 연산자의 동작은 함수 본체에 컴파일러가 코드를 직접 추가함
- 암시적 복사 대입 연산자
  - 부모 클래스에 대해 복사 대입 연산자 호출
  - 멤버가 클래스 타입일 경우 복사 대입 연산자 호출
  - 멤버가 기본 타입일 경우 메모리 복사
  - 멤버가 배열 타입일 경우 원소의 타입에 따라 복사 대입 연산자 호출 또는 메모리 복사 수행
- 명시적 복사 대입 연산자
  - 프로그래머가 정의한 동작 실행

# 클래스 : 초기화 리스트

- 생성자가 호출될 때 선처리 영역에 미리 정의된 것을 수행하지 않고 초기화 리스트에서 지시한 내용을 대신 수행함
  - > 단, 멤버 변수의 경우 생성자내에서 초기화 하나 초기화 리스트를 사용하나 생성되는 어셈블리코드는 동일함
- 초기화 리스트가 필요한 경우
  - 상수 타입 멤버 초기화
  - 레퍼런스 멤버 초기화
  - 부모 클래스 기본 생성자가 아닌 생성자 호출
- 초기화 리스트 사용 주의점
  - 자식 클래스에서 부모 클래스 멤버를 초기화 하기 위해서는 부모 클래스의 생성자를 정의해 호출해야 함

# 클래스 : 상속

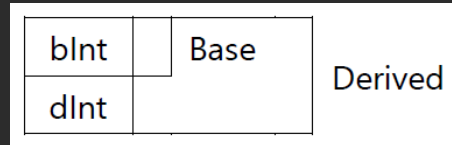
- 상속된 클래스의 메모리 구조

```
#include <iostream>

using namespace std;

class Base
{
    int blnt;
};

class Derived : public Base
{
    int dInt;
};
```



- 상속이 되지 않는 것 : 생성자, 소멸자, 대입연산자
- 상속이 되는 것 : 타입 변환 연산자, 일반 연산자
- 가상 상속 : 중복제거

# 함수 : 함수 호출 규약

- 함수 호출 규약 : 어떻게 서브루틴이 그들의 호출자(caller)로부터 변수를 받고, 어떻게 결과물을 반환하는지에 대한 규약
- 함수 호출 규약 종류

Segment word size	Calling Convention	Parameters in registers	Parameter order on stack	Stack clean by
32bit	__cdecl		C	Caller
	__stdcall		C	Function
	__fastcall	ecx,edx	C	Function
	__thiscall	ecx	C	Function
64bit	Windows	rcx/xmm0,	C	Caller
		rdx/xmm1,	C	
		r8/xmm2,	C	
		r9/xmm3	C	
	Linux,BSD (GNU,Intel)	rdi,rsi	C	Caller
		rdx,rcx,r8	C	
		r9,xmm0-7	C	

# 함수 : 함수 호출 규약

- \_cdecl의 스택 프레임

```
#include <iostream>

using namespace std;

int Plus(int a, int b);

int main()
{
    int sum = Plus(1, 2);
}

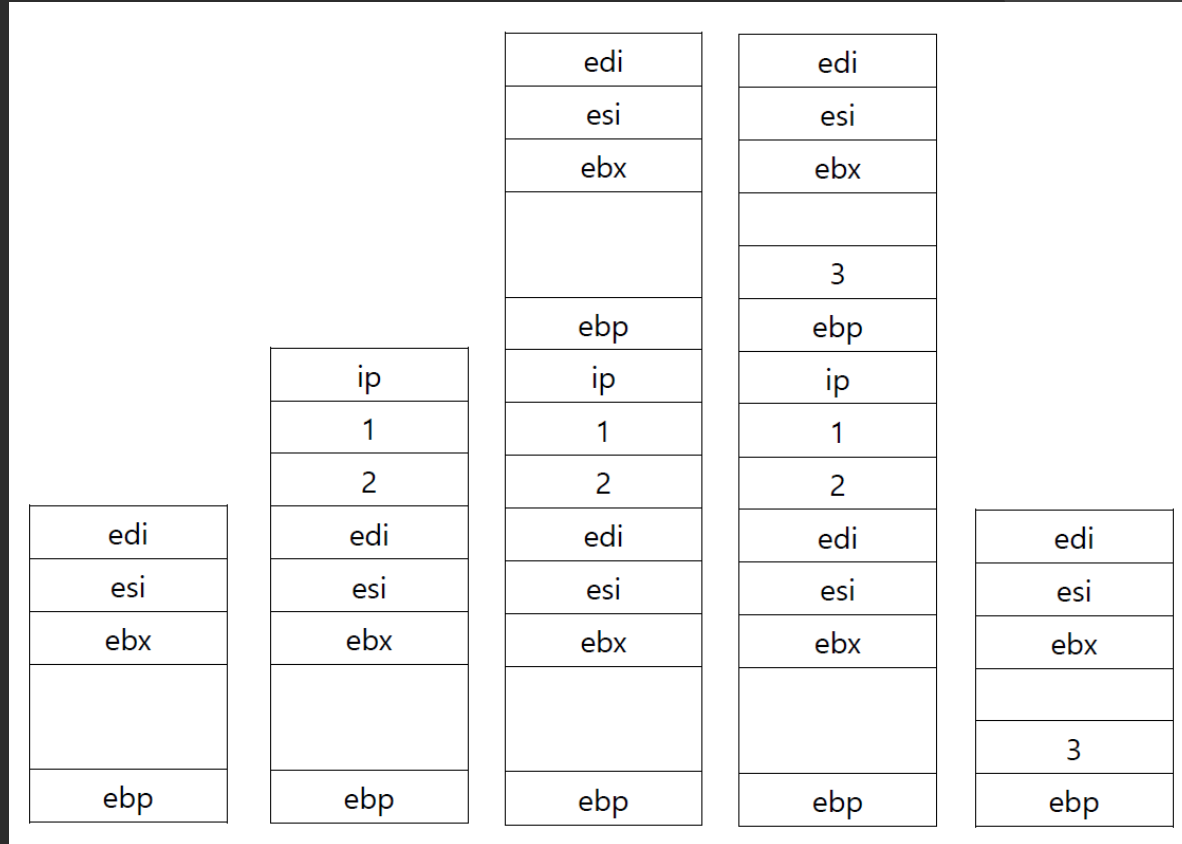
int Plus(int a, int b)
{
    return a + b;
}
```

```
int main()
{
000E1DF0 push    ebp
000E1DF1 mov     ebp,esp
000E1DF3 sub     esp,000Ch
000E1DF9 push    ebx
000E1DFA push    esi
000E1DFB push    edi
000E1DFC lea     edi,[ebp-000Ch]
000E1E02 mov     ecx,33h
000E1E07 mov     eax,00000000h
000E1E0C rep stos dword ptr es:[edi]
000E1E0E mov     ecx,offset _48B431D3_소스@cpp (0F1027h)
000E1E13 call   @__CheckForDebuggerJustMyCode@4 (0E133Eh)
        int sum = Plus(1, 2);
000E1E18 push    2
000E1E1A push    1
000E1E1C call   Plus (0E1627h)
000E1E21 add     esp,8
000E1E24 mov     dword ptr [sum],eax
}
000E1E27 xor     eax,eax
000E1E29 pop     edi
000E1E2A pop     esi
000E1E2B pop     ebx
000E1E2C add     esp,000Ch
000E1E32 cmp     ebp,esp
000E1E34 call   __RTC_CheckEsp (0E1357h)
000E1E39 mov     esp,ebp
}
000E1E3B pop     ebp
000E1E3C ret
```

```
int Plus(int a, int b)
{
000E1D10 push    ebp
000E1D11 mov     ebp,esp
000E1D13 sub     esp,000Ch
000E1D19 push    ebx
000E1D1A push    esi
000E1D1B push    edi
000E1D1C lea     edi,[ebp-000Ch]
000E1D22 mov     ecx,33h
000E1D27 mov     eax,00000000h
000E1D2C rep stos dword ptr es:[edi]
000E1D2E mov     ecx,offset _48B431D3_소스@cpp (0F1027h)
000E1D33 call   @__CheckForDebuggerJustMyCode@4 (0E133Eh)
        int sum = a + b;
000E1D38 mov     eax,dword ptr [a]
000E1D3B add     eax,dword ptr [b]
000E1D3E mov     dword ptr [sum],eax
        return sum;
000E1D41 mov     eax,dword ptr [sum]
}
000E1D44 pop     edi
000E1D45 pop     esi
000E1D46 pop     ebx
000E1D47 add     esp,000Ch
000E1D4D cmp     ebp,esp
000E1D4F call   __RTC_CheckEsp (0E1357h)
000E1D54 mov     esp,ebp
000E1D56 pop     ebp
000E1D57 ret
```

# 함수 : 함수 호출 규약

- \_cdecl의 스택 프레임
  - 디버그 모드 한정



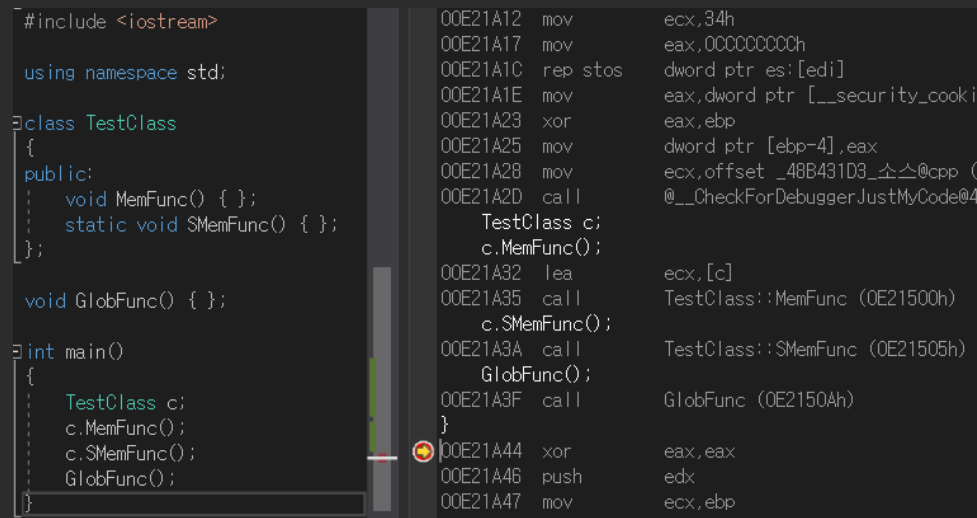
# 함수 : 네임 데코레이션

- C++의 경우 함수 중복정의(함수 오버로딩)이 가능
  - > 내부적으로 C와는 다른 함수 이름 규칙을 사용, 생성하는 내부 이름은 컴파일러마다 다름
- C컴파일러
  - 이름을 기반으로 컴파일러가 내부적으로 사용하는 이름을 생성
- C++컴파일러
  - 함수 이름과 인자 개수, 타입정보를 기반으로 내부적으로 사용하는 이름 생성
  - `extern "C"` : 네임 데코레이션을 할때 C방식으로 하라는 의미
    - > 만약 이것을 사용하지 않고 C로 컴파일된 소스를 가져오려 한다면 링크에러



# 함수 : 클래스 멤버 함수

- 클래스 멤버 함수도 전역 함수들처럼 Text(Code)영역에 존재함
- 비정적 클래스 멤버 함수의 호출 규약 : thiscall (32bit x86기준)
  - 스택이 아닌 ecx(32bit), rcx(64bit)레지스터를 이용해 this(자기 자신을 가리키는 포인터)를 넘김  
-> 비정적 클래스의 멤버 함수 호출시에는 자기 자신에 대한 정보가 포함됨



```
#include <iostream>

using namespace std;

class TestClass
{
public:
    void MemFunc() { };
    static void SMemFunc() { };
};

void GlobFunc() { };

int main()
{
    TestClass c;
    c.MemFunc();
    c.SMemFunc();
    GlobFunc();
}
```

```
00E21A12 mov     ecx,34h
00E21A17 mov     eax,00000000h
00E21A1C rep stos dword ptr es:[edi]
00E21A1E mov     eax,dword ptr [__security_cookie]
00E21A23 xor     eax,ebp
00E21A25 mov     dword ptr [ebp-4],eax
00E21A28 mov     ecx,offset _40B431D3_소스@cpp (00401000)
00E21A2D call    @__CheckForDebuggerJustMyCode@4
TestClass c;
c.MemFunc();
00E21A32 lea     ecx,[c]
00E21A35 call    TestClass::MemFunc (0E21500h)
c.SMemFunc();
00E21A3A call    TestClass::SMemFunc (0E21505h)
GlobFunc();
00E21A3F call    GlobFunc (0E2150Ah)
}
00E21A44 xor     eax,eax
00E21A46 push    edx
00E21A47 mov     ecx,ebp
```

# 함수 : 클래스 멤버 함수

- 이름 탐색 규칙
  - 하위 클래스에서 상위 클래스순으로 검색
  - 이름으로 탐색, 그러므로 오버라이딩된 함수는 탐색 범위에서 가장 가까운 것만 검색
- const 멤버 함수
  - const객체는 const멤버 함수만 호출할 수 있다
  - const멤버 함수는 객체의 상태를 변경시킬 수 없다
  - const도 함수의 시그니처에 포함된다
  - STL호환 클래스 작성시 사용해야 할 경우도 있음

```
#include <iostream>

using namespace std;

class Base
{
public:
    void FuncB() { };
    void Func0(int i) { };
    void Func02(int i) { };
};

class Derived : public Base
{
public:
    void FuncD() { };
    void Func0() { };
    void Func02(double i) { };
};

int main()
{
    Derived d;
    d.FuncB();
    d.FuncD();
    //d.Func0(1); //컴파일 에러
    d.Base::Func0(1);
    d.Func02(1); //Derived에 있는 함수 호출
}
```

# 함수 : 가상 함수

- 가상함수의 동작 : 실제 타입으로 변환할 필요 없이 인터페이스 타입으로 실제 타입의 메소드를 불러올 수 있음, 가상함수 테이블을 참조해 실제 타입의 메소드를 찾음

```
#include <iostream>

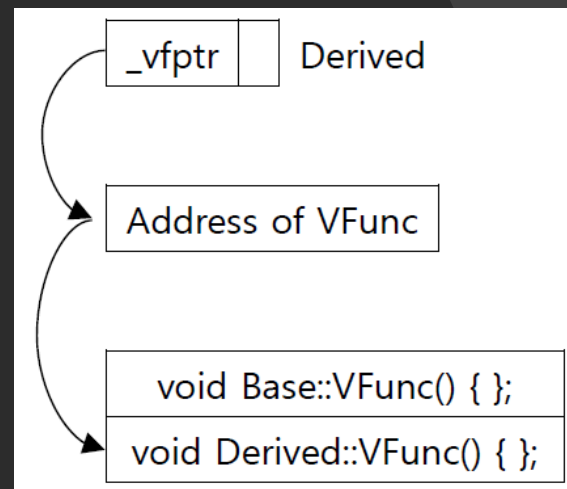
using namespace std;

class Base
{
public:
    virtual void VFunc() { };
};

class Derived : public Base
{
public:
    virtual void VFunc() { };
};

int main()
{
    Base *pBase = new Derived();
    pBase->VFunc();
    delete pBase;
}
```

```
000C1C49 mov     dword ptr [ecx],eax
000C1C4B mov     ecx,dword ptr [ebp-0D4h]
000C1C51 call    Derived::Derived (0C1537h)
000C1C56 mov     dword ptr [ebp-0E8h],eax
000C1C5C jmp     main+68h (0C1C68h)
000C1C5E mov     dword ptr [ebp-0E8h],0
000C1C68 mov     edx,dword ptr [ebp-0E8h]
000C1C6E mov     dword ptr [pBase],edx
                pBase->VFunc();
000C1C71 mov     eax,dword ptr [pBase]
                pBase->VFunc();
000C1C74 mov     edx,dword ptr [eax]
000C1C76 mov     esi,esp
000C1C78 mov     ecx,dword ptr [pBase]
000C1C7B mov     eax,dword ptr [edx]
000C1C7D call    eax
000C1C7F cmp     esi,esp
000C1C81 call    __RTC_CheckEsp (0C12E9h)
                delete pBase;
000C1C86 mov     eax,dword ptr [pBase]
000C1C89 mov     dword ptr [ebp-0E0h],eax
000C1C8F push    4
```



# 템플릿 : 템플릿

- 함수 템플릿

```
template<typename T>  
void Func(T type)  
{ }
```

- 클래스 템플릿

```
template<typename T>  
class TClass  
{  
private:  
    T data;  
public:  
    TClass(T data) { }  
};
```

- 템플릿은 컴파일 타임에 호출될 타입이 결정되면 실체가 생성됨

# 코드 리뷰

1주차 과제