

2장 더 나은 C로서의 C++ (1)

- ✦ 표준 입출력
- ✦ 네임스페이스 (고전 C++와 표준 C++)
- ✦ 함수 오버로딩
- ✦ 디폴트 매개변수
- ✦ new와 delete
- ✦ bool 자료형
- ✦ C++ is not C

1. 표준 입출력

✦ C++ : C의 모든 라이브러리를 포함

- printf, scanf 함수 사용 가능

✦ 예 : int, double, 문자열 값을 입력받고 출력하기

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

int main(void)
{
    int iVar;
    double dVar;
    char str[20];

    printf("int, double, 문자열 입력 : ");
    scanf("%d %lf %s", &iVar, &dVar, str);

    printf("int 값 : %d\n", iVar);
    printf("double 값: %f\n", dVar);
    printf("문자열 : %s\n", str);

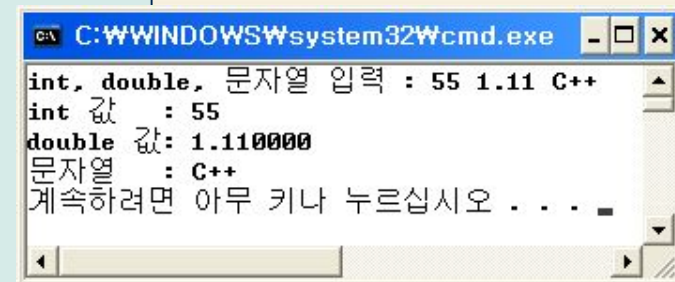
    return 0;
}
```

← cstdio : ??

← VC++ 6.0의 경우 삭제

→ 2.2 네임스페이스 참고

← 주소 전달

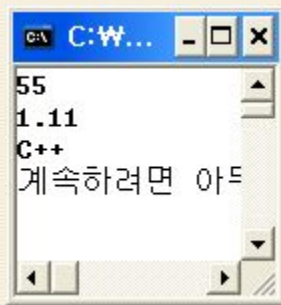


1. 표준 입출력 : cin, cout의 사용

C++의 표준 입출력 방법

- cin : 표준 입력 객체, >> 연산자와 함께 사용
- cout : 표준 출력 객체, << 연산자와 함께 사용

예 : cout



```
#include <iostream>           // cin, cout 표준 입출력 포함
using namespace std;          // 반드시 들어가야 함

int main(void)
{
    int iVar = 55;
    double dVar = 1.11;
    char str[20] = "C++";

    cout << iVar << endl;      // 표준 출력 연산자 << 사용
    cout << dVar << endl << str << endl;

    return 0;
}
```

연속적인 << 사용 가능
더 이상 타입에 신경쓰지 않아도 됨!

endl → “\n”과 동일.
바꾸어 수행해 보라.

1. 표준 입출력 : cin, cout의 사용

예 : cin, cout의 사용

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

```
int main(void)
{
```

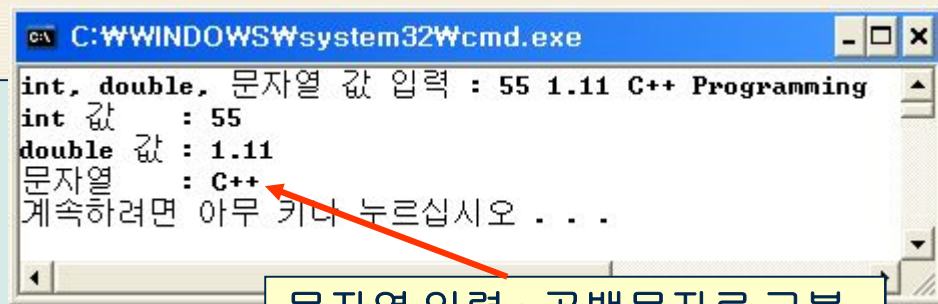
```
    int iVar;
    double dVar;
    char str[20];
```

```
    cout << "int, double, 문자열 값 입력 : ";
    cin >> iVar >> dVar;           // 표준 입력 연산자 >> 사용
    cin >> str;
```

```
    cout << "int 값    : " << iVar << endl;
    cout << "double 값: " << dVar << endl;
    cout << "문자열   : " << str << endl;
```

```
    return 0;
```

```
}
```



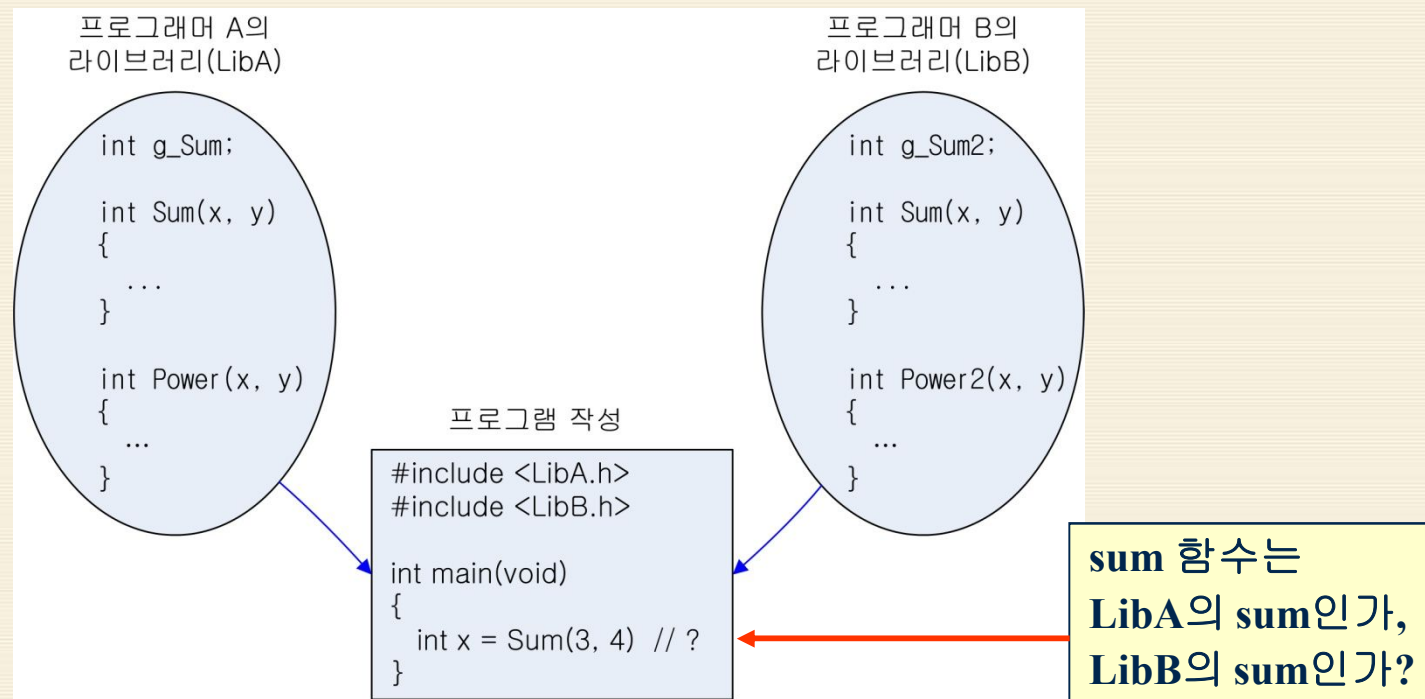
문자열 입력 : 공백문자로 구분
줄 단위 입력 → 11장

연속적인 >> 사용 가능
더 이상 타입에 신경쓰지 않아도 됨!

2. 네임스페이스

▣ 다음 시나리오의 문제점은?

- 프로그래머A와 프로그래머B가 만든 라이브러리를 사용하여 프로그램을 작성하고자 한다.



- 문제점 : 이름(함수명, 변수명 등)의 충돌 가능성이 있음
- 해결 방안 : 네임스페이스(이름공간)

2. 네임스페이스

≡ 네임스페이스 : 식별자들에 대한 그룹

≡ 예 : Microsoft, Samsung 네임스페이스 작성

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

Microsoft 네임스페이스

```
namespace Microsoft {
    int g_MVar;
    int Plus(int x, int y)
    {
        return (x + y);
    }
    int Minus(int x, int y)
    {
        return (x - y);
    }
}
```

```
namespace Samsung {
    int g_SVar;
    int Plus(int x, int y)
    {
        return (x + y);
    }
    int Minus(int x, int y);
}
```

Samsung 네임스페이스

둘 다 Plus, Minus 존재

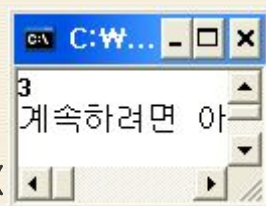
```
int Samsung::Minus(int x, int y)
{
    return (x - y);
}
```

함수의 외부 정의 :: 사용

```
int main(void)
{
    Microsoft::g_MVar = Microsoft::Minus(5, 2);
    cout << Microsoft::g_MVar << endl;

    return 0;
}
```

네임스페이스 이름을 통해 구별



2. 네임스페이스

⌘ 전역 네임스페이스

- 어떤 네임스페이스에도 포함되어 있지 않는 영역
- 예 : 전역 네임스페이스에 Sum 함수가 있는 경우
 - Sum(3, 4) 또는 ::Sum(3, 4)

⌘ 특정 네임스페이스에 포함되어 있는 식별자를 전역 네임스페이스(또는 다른 네임스페이스)에 있는 것처럼 사용하는 방법

- using Microsoft::Minus;
- using Samsung::Plus;

⌘ 특정 네임스페이스에 포함되어 있는 모든 식별자를 전역 네임스페이스에 있는 것처럼 사용하는 방법

- using namespace Microsoft;
- using namespace Samsung;

⌘ 단, 중복 선언되어 있는 식별자의 경우 여전히 해당 네임스페이스명을 붙여야 함!

2. 네임스페이스 : 고전 C++와 표준 C++

■ 고전 C++ : 네임스페이스의 개념이 없음

```
#include <iostream.h> // 고전 C++ 헤더 파일

int main(void)
{
    cout << "고전 C++" << endl;

    return 0;
}
```

VC++ 6.0 지원함
VC++ 8.0 지원하지 않음

■ 표준 C++ : library의 모든 식별자 → std 네임스페이스

```
#include <iostream>

using std::cout;
using std::endl;

int main(void)
{
    cout << "표준 C++" << endl;
    // std::cout << "표준 C++" << std::endl; // using 선언이 없을 경우

    return 0;
}
```

using namespace std;
모든 식별자 사용 가능!

표준 C++의 헤더 파일
VC++ 6.0, 8.0 모두 지원

2. 네임스페이스 : 고전 C++와 표준 C++

■ 기존 C 함수의 사용 ← 실제로 C 라이브러리를 사용함

```
#include <stdio.h>

int main(void)
{
    printf("C++ Programming");
    return 0;
}
```

다음과 같이 대체 가능
#include <cstdio>
using namespace std;

원리

cstdio 파일은 다음과 같이 **stdio.h**를 **include**하고 있으며, **stdio.h** 내의 모든 식별자를 **std** 네임스페이스로 포함시키고 있음

using 선언이란 특정 네임스페이스의 식별자를 다른 네임스페이스에 위치시키는 것!

모든 C 라이브러리들이 이와 같은 방식으로 사용되고 있음

주의 : VC++ 6.0의 경우 **cstdio** 파일에는 단순히 **#include <stdio.h>**만 기술되어 있음
따라서, **std**에 포함되지 않으며 **using ...** 사용 불가

```
// cstdio 파일
#include <stdio.h>

namespace std {
    using ::printf;
    using ::scanf;
    using ::fread;
    using ::fwrite;
    using ::FILE;
    ..... // 생략
}
```

전역에 있는 **printf**를 **std** 네임스페이스에 포함시킴

2. 네임스페이스 : 중첩 네임스페이스

✦ 예 : CompanyA 네임스페이스 내에 DeptC라는 네임스페이스 선언

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

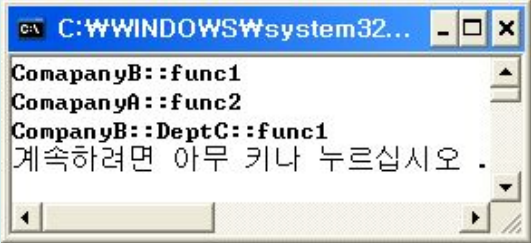
namespace CompanyA {                // 네임스페이스 CompanyA 작성
    int g_VarA;
    void func1() { cout << "ComapanyA::func1" << endl; }
    void func2() { cout << "ComapanyA::func2" << endl; }
}

namespace CompanyB {                // 네임스페이스 CompanyB 작성
    using namespace CompanyA;        // 네임스페이스 CompanyA를 CompanyB로 포함
    int g_VarB;
    void func1() { cout << "ComapanyB::func1" << endl; }

    namespace DeptC {                // CompanyB 내에 네임스페이스 DeptC 작성
        void func1() { cout << "CompanyB::DeptC::func1" << endl; }
    }
}

int main(void)
{
    CompanyB::func1();
    CompanyB::func2();                // 실제로는 CompanyA의 func2 함수실행
    CompanyB::DeptC::func1();

    return 0;
}
```



3. 함수 오버로딩

예 : 다음 프로그램의 문제점은?

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

void swap(int *x, int *y)           // 실매개변수 2개의 값 교환
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main(void)
{
    int a = 3, b = 4;
    double c = 1.1, d = 2.2;

    swap(&a, &b);
    swap(&c, &d);

    cout << "a = " << a << ", b = " << b << endl;
    cout << "c = " << c << ", d = " << d << endl;

    return 0;
}
```

컴파일 에러 : **double * → int *** 묵시적 형변환 불가
swap((int *) &c, (int *) &d)로 변경한다면?
논리적 에러 : 소수점 이하 처리 불가
해결책 : **int, double형 별도의 swap 함수 작성!**

3. 함수 오버로딩

swap 함수의 수정

```
void swapi(int *x, int *y)           // int형 값 2개 교환
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
void swapd(double *x, double *y)    // double형 값 2개 교환
{
    double temp = *x;
    *x = *y;
    *y = temp;
}
```

문제점 : 호출하는 곳에서도
swapi(&a, &b);
swapd(&c, &d);
로 변경
* 동일한 함수명 사용 가능
→ 함수 오버로딩

```
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
void swap(double *x, double *y)
{
    double temp = *x;
    *x = *y;
    *y = temp;
}
```

3. 함수 오버로딩

■ 함수 오버로딩의 조건

- 함수명 또는 매개변수(개수 또는 타입)가 달라야 함
- 반환형은 무관
 - `int func(int n);` `char func(int n);` // 오버로딩 불가능

■ 함수 오버로딩 시 모호성 주의

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

char square(char c)                // char 값 제공
{
    return (c * c);
}

long square(long val)              // long 값 제공
{
    return (val * val);
}

int main(void)
{
    cout << square(3) << endl;    // 3(int)은 char, long 둘 다로 형변환 가능
    return 0;
}
```

컴파일 에러 발생

4. 디폴트 매개변수

- 예 : x의 y승을 구하는 power 함수 작성. 단, 두 번째 매개변수 y가 전달되지 않으면 x의 2승을 반환

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

int power(int x)                // x의 2승
{
    return (x * x);
}

int power(int x, int y)        // x의 y승
{
    int i;
    int result = 1;

    for (i = 0; i < y; i++)
        result *= x;

    return result;
}

int main(void)
{
    cout << power(3) << endl;    // 3의 2승
    cout << power(3, 3) << endl; // 3의 3승

    return 0;
}
```

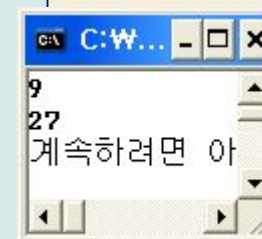
함수 오버로딩을 통해 작성 가능

디폴트 매개변수를 사용한다면
하나의 함수로 작성 끝!

```
int power(int x, int y = 2)
{
    int i;
    int result = 1;

    for (i = 0; i < y; i++)
        result *= x;

    return result;
}
```



4. 디폴트 매개변수

⌘ 디폴트 매개변수의 복잡한 사용 예

- `int power(int a, int b);`
- `int f(int x, int y = power(3, 4));` // 디폴트 값 : 함수 호출 가능

⌘ 주의 사항

- 디폴트 매개변수 값은 뒤에서부터 줄 수 있음
 - 즉, 함수 호출 시 마지막 값부터 역순으로 생략 가능
 - `int f(int a, int b = 3, int c = 5);` `f(5);` // O
 - `int g(int a, int b = 3, int c);` `g(1, , 3);` // X
- 디폴트 매개변수 값은 함수 프로토타입 또는 정의 둘 중 하나에만 기술
→ 일반적으로 프로토타입에 기술
 - `int f(int a = 3);` `int f(int a = 3) { return a; }` // X
- 함수 오버로딩과 함께 사용할 경우 모호성 주의
 - `int power(int x, int y = 2);`
 - `int power(int x);`
 - `power(3);` // 둘 다 호출 가능, 어떤 함수?

5. new와 delete

C

- malloc과 free를 사용한 동적 할당 및 해제
- `int *p1 = (int *) malloc(sizeof(int));` `free(p1);`
- `int *p2 = (int *) malloc(sizeof(int) *3);` `free(p1);`

C++

- malloc, free 사용 가능
- new, delete 연산자 사용
 - malloc, free 보다 편리
 - `int *p1 = new int;` `delete p1;`
 - `int *p2 = new int[3];` `delete [] p2;` // 배열 메모리 해제
- 메모리 할당과 동시에 초기화 가능
 - `int *p1 = new int(100);` // 배열의 경우 초기화 불가능

5. new와 delete

예 : int형 변수 하나의 동적 할당 및 초기화

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

```
int main(void)
{
```

```
    int *p = new int(100);    // int형 변수 동적 할당 및 100으로 초기화
```

```
    cout << "포인터 변수의 주소 : " << &p << endl;
```

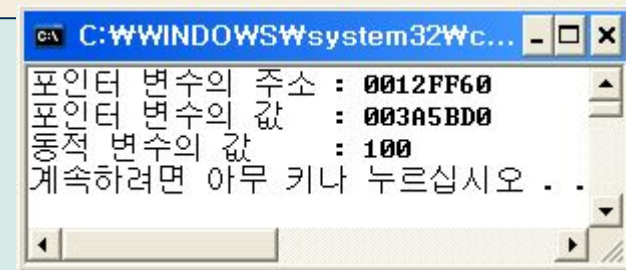
```
    cout << "포인터 변수의 값 : " << p << endl;
```

```
    cout << "동적 변수의 값 : " << *p << endl;
```

```
    delete p;
```

```
    return 0;
```

```
}
```



5. new와 delete

예 : 배열의 동적 할당

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

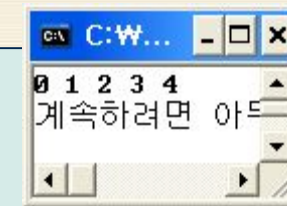
int main(void)
{
    int i;
    int *p = new int[5];    // int형 변수 5개 동적 할당

    for (i = 0; i < 5; i++)
        p[i] = i;          // 포인터를 통해 배열처럼 사용

    for (i = 0; i < 5; i++)
        cout << p[i] << " ";
    cout << endl;

    delete [] p;           // 배열 동적 할당 시 해제

    return 0;
}
```



5. new와 delete

예 : 2차원 배열의 동적 할당

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

```
int main(void)
{
```

```
    int **p;
```

```
    int i, j;
```

```
    p = new int *[4];
```

```
    for (i = 0; i < 4; i++)
        p[i] = new int[5];
```

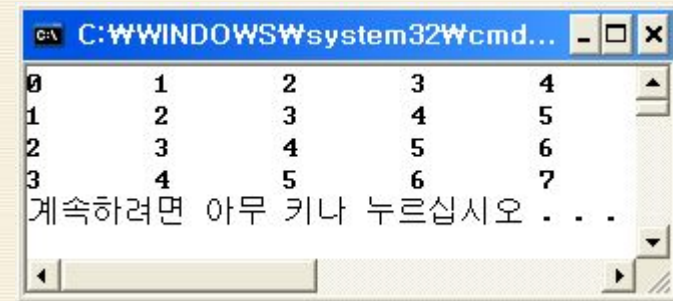
```
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 5; j++) {
            p[i][j] = i + j;
        }
    }
```

2차원 배열처럼 사용

int형 포인터에 대한 포인터

p[0], ..., p[3]이 int형 포인터

각 포인터에 대해
int형 5개 배열 생성



```
        for (i = 0; i < 4; i++) {
            for (j = 0; j < 5; j++) {
                cout << p[i][j] << "Wt";
            }
            cout << endl;
        }
```

```
    for (i = 0; i < 4; i++)
        delete [] p[i];
    delete [] p;
```

```
    return 0;
```

```
}
```

동적 메모리 해제

6. bool 자료형

▣ C 스타일의 bool 자료형

- typedef을 통해 참, 거짓을 표시하는 타입 선언, 내부적으로 int형
- typedef int BOOL;

▣ C++ : bool형 도입

- 값 : true, false
- 수식 연산에 사용 가능
 - true → 1, false → 0
- 다른 타입의 값을 bool형으로 자동 형변환 가능
 - 0 → false, 그 외 → true
 - ✓ bool tf1 = true;
 - ✓ int a = 1 - tf1 // a는 0이 됨
 - ✓ bool tf2 = a; // tf2는 false가 됨

6. bool 자료형

예 : C, C++의 비교

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

```
typedef int BOOL;
```

C 스타일

```
int main(void)
{
```

```
    BOOL bVar1 = 3;
    bool bVar2 = 3;
```

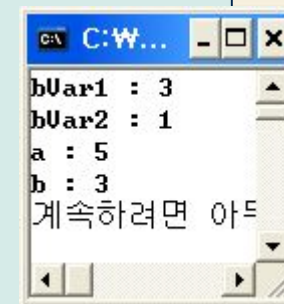
```
    cout << "bVar1 : " << bVar1 << endl;
    cout << "bVar2 : " << bVar2 << endl;
```

```
    int a = bVar1 + 2;           // BOOL은 int형으로 취급됨
    int b = bVar2 + 2;           // bool은 true, false로 취급됨
```

```
    cout << "a : " << a << endl;
    cout << "b : " << b << endl;
```

```
    return 0;
```

```
}
```



7. C++ is not C

C와 C++ 사이의 차이 (몇 가지 예)

1. void 포인터와 다른 타입의 포인터 사이의 자동 형변환
2. 지역 변수의 선언 위치
3. 함수 프로토타입에서 매개변수가 없는 경우의 해석
4. 구조체 변수의 선언 방법

1. void 포인터 \leftrightarrow 다른 타입의 포인터 자동 형변환

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

```
int main(void)
{
    int *p = malloc(sizeof(int) * 3);
    free(p);
    return 0;
}
```

C에서는 가능 : 자동 형변환 O

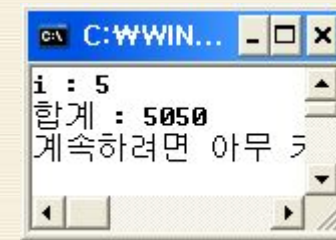
int *p = malloc(sizeof(int) * 3); // (void *) 반환, C++ 자동 형변환 X

int *p = (int *) malloc(sizeof(int) * 3);

7. C++ is not C

2. 지역 변수의 선언 위치

- C : 다른 문장이 나오기 이전
- C++ : 중간에도 선언 가능



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

```
int main(void)
{
```

```
    int i = 5;
    int Sum;
    Sum = 0;
```

C++ : for문에서도 선언 가능, 두 개의 i는 서로 다른 변수

```
    for (int i = 1; i <= 100; i++) { // i는 for 블록 내에서만 사용 가능
        Sum += i;
    }
```

```
    cout << "i : " << i << endl;
    cout << "합계: " << Sum << endl;
```

```
    return 0;
```

```
}
```

주의 : VC++ 6.0에서는 for문에서 선언한 변수를 그 이후에 계속 사용 가능 → 컴파일 에러 발생
→ 두 개의 i를 같은 변수로 봄
→ 표준 C++의 개념에 맞지 않음

7. C++ is not C

3. 함수의 매개변수가 없는 경우의 해석

- C : 해당 함수 호출 시 실매개변수의 개수, 타입을 검사하지 않음
→ 실매개변수에 관계없이 호출 가능 (실매개변수 무시)
- C++ : void로 간주
 - void f(); // void f(void);와 동일

```
#include <stdio.h>

void f() {                    // 매개변수를 지정하지 않은 경우
    printf("test\n");
}

int main(void)
{
    f(3.5);
    f(3);
    f('a');
    f(1, 2);

    return 0;
}
```

C에서는 수행 가능

C++에서는 수행 불가능!
f 함수 호출 시 실매개변수가 없어야 함

7. C++ is not C

4. 구조체 변수의 선언 방법

- C : struct Point P1; // struct 반드시 동반
- C++ : struct Point P1; Point P1; // struct 없이도 가능

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

```
struct Point {
    int x, y;
};
```

```
int main(void)
{
```

```
    Point P1 = { 1, 2 }; // struct를 포함하지 않고 변수 선언
```

```
    return 0;
```

```
}
```

C에서는 에러 발생

※ C, C++ 모두 끊임없이 진화 : 서로의 장점 참고