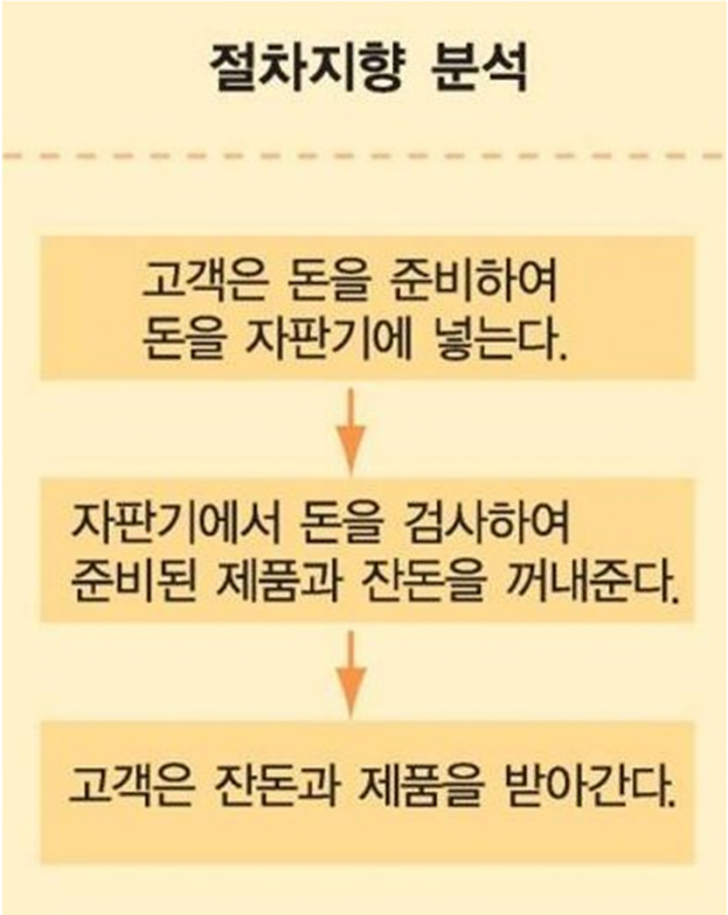
C vs. C++

➢ C: 절차적(procedural) 프로그래밍 언어

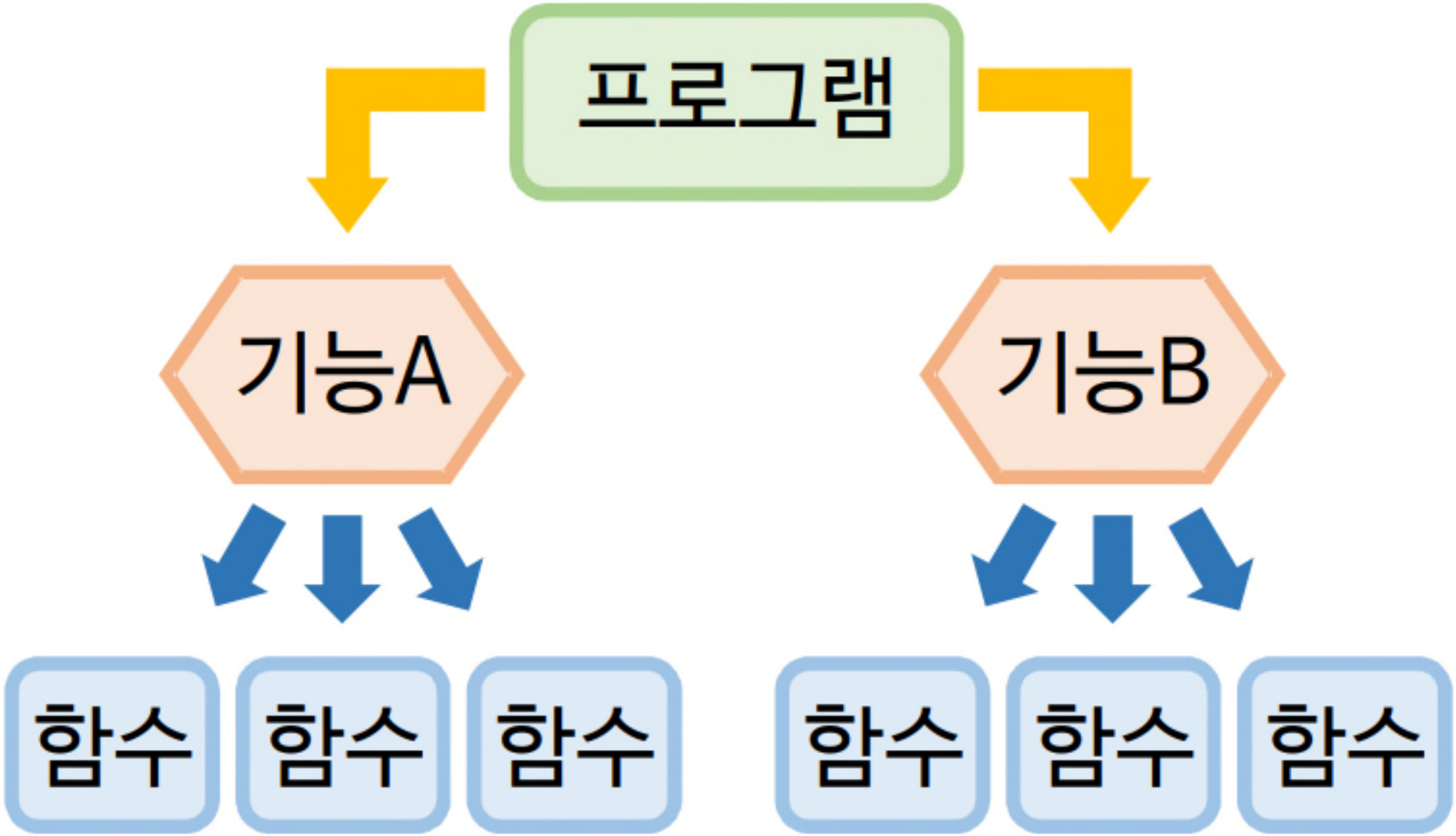
- 프로시저(procedure)를 이용하여 작성하는 프로그래밍(하향식 설계)
  - 프로시저: 루틴(i.e. main function), 서브루틴(i.e. main문 밖에서 정의한 코드블럭 중 반환 없는 것), 함수(i.e. main문 밖에서 정의한 코드 블럭 중 반환 값이 있는 것)를 의미함
  - 프로그램 진행에 따라 명령어/함수호출을 통한 계산 수행 및 결과를 초기 설정한 자료구조에 저장
- main()함수에서 반복문/조건문 등을 통해 흐름제어
  - 알고리즘 개발이 중요함(구조적 프로그래밍)
  - 과한 중첩/잦은 이동은 좋은 코드가 아님
- 프로그램이 어느 시점에 어떤 일을 처리해야하는지 설계하고 구현하는 것이 중요함



C vs. C++

➢ C: 절차적(procedural) 프로그래밍 언어

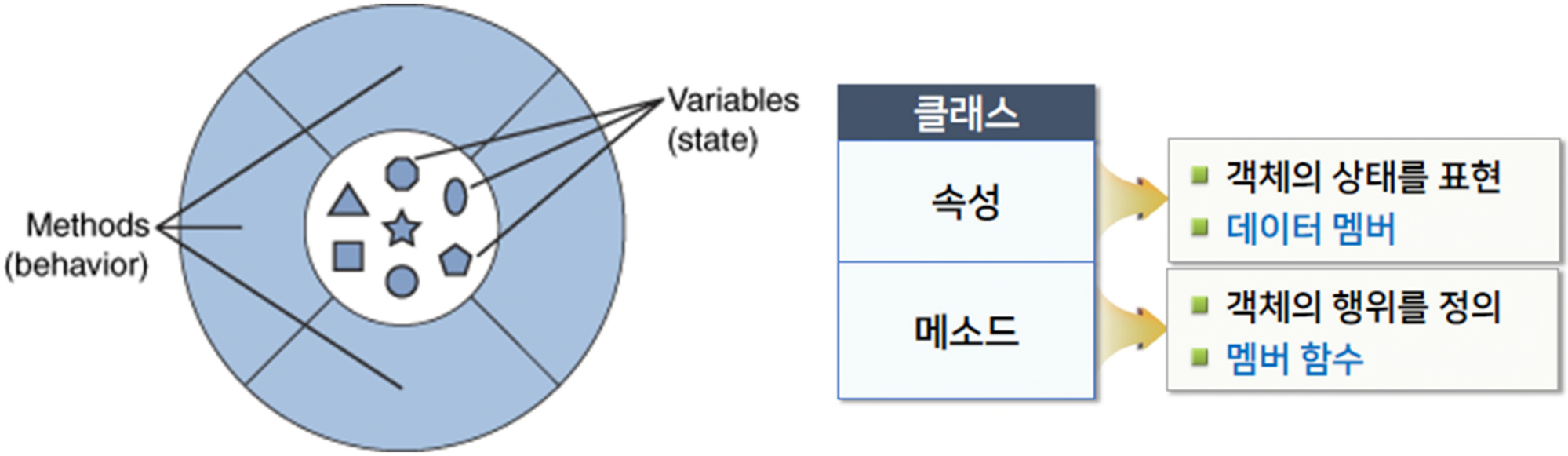
- 출처:천재학습백과, '구조적 프로그래밍' http://koc.chunjae.co.kr/Dic/dicDetail.do?idx=43508



C vs. C++

➢ C++: 객체지향 프로그래밍 언어 (C with Classes)

- 해결해야 하는 문제의 특성에 맞는 데이터 설계가 중요
  - 프로그래머가 생각하는 기능을 포함하여 설정한 기준에 따른 객체를 중심으로 설계
    - 목적에 따라서 관련 코드와 데이터를 적절하게 구분하고 사용할 수 있도록 객체를 개발해야 함
  - 클래스(class): 변수/함수를 포함할 수 있는 일종의 구조체
    - 기본적인 접근 지정자의 차이 (C의 구조체: public, C++의 클래스: private)
    - 해당 클래스의 객체를 생성하여 접근하고 사용할 수 있음 (i.e. 메모리가 할당됨)

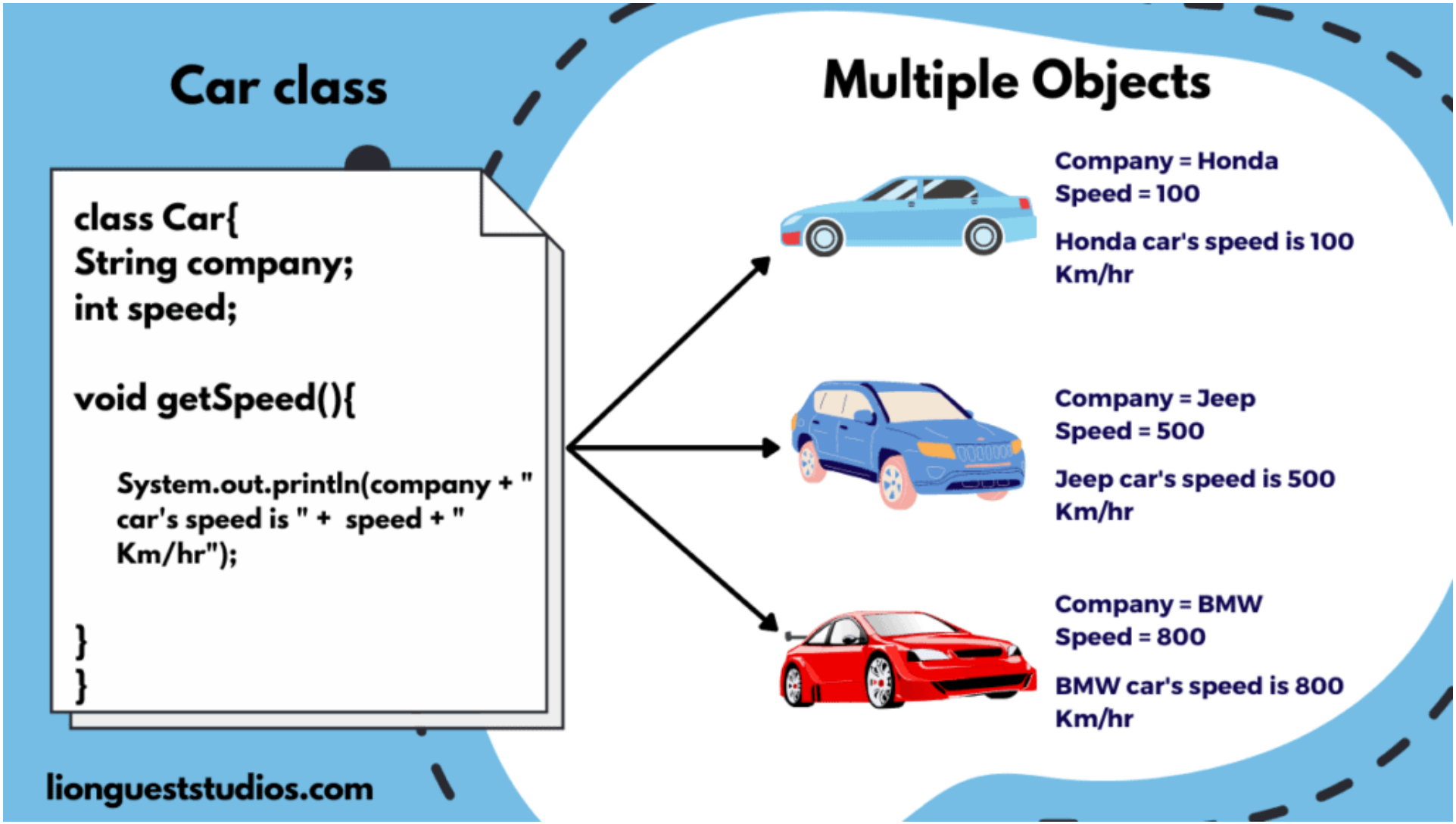


C++의 객체와 관련된 가장 기본적인 개념도

C vs. C++

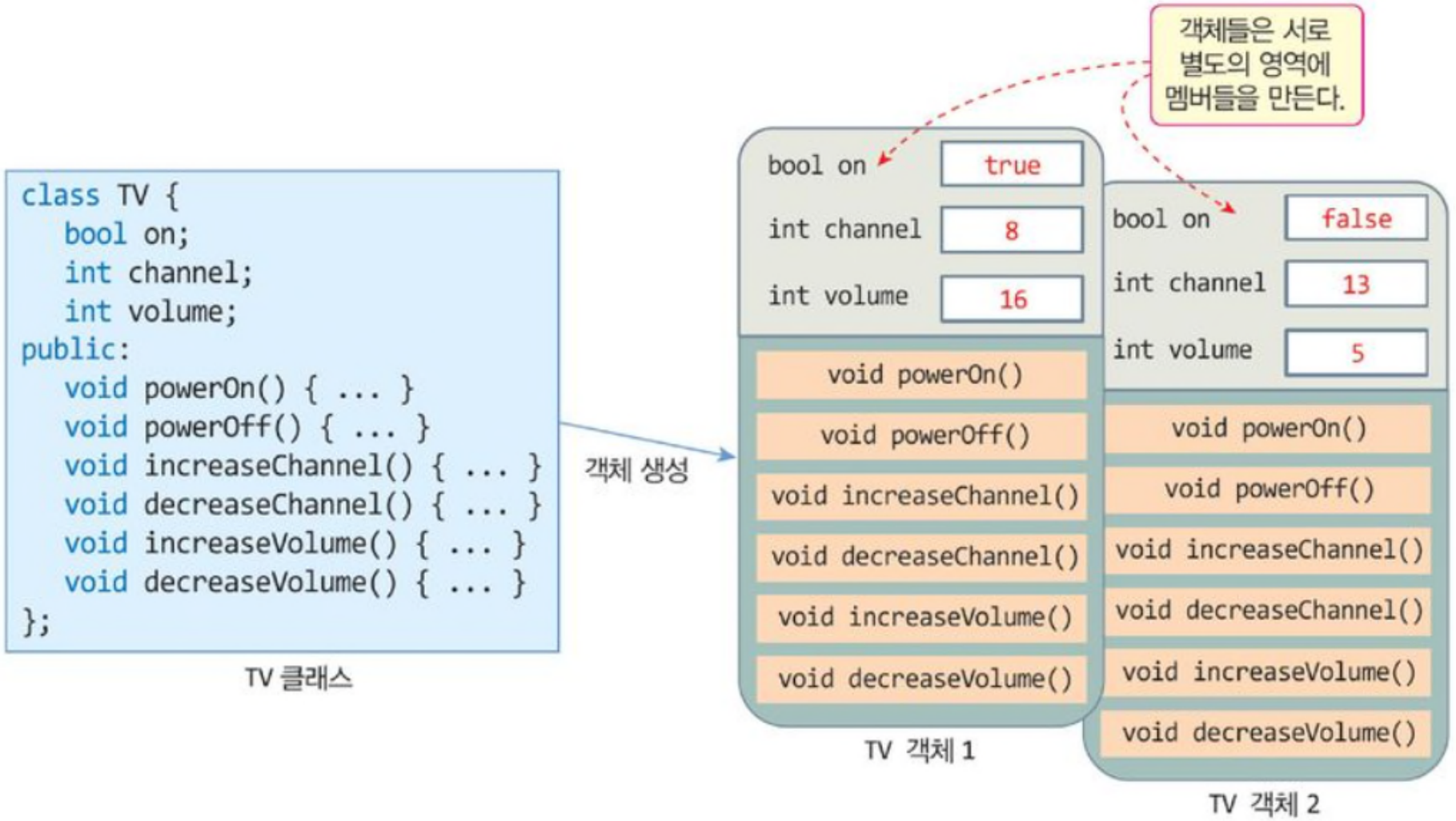
➢ C++: 객체지향 프로그래밍 언어

- 객체간의 상호작용을 통해 원하는 기능을 구현할 수 있음



➢ C++: 객체지향 프로그래밍 언어

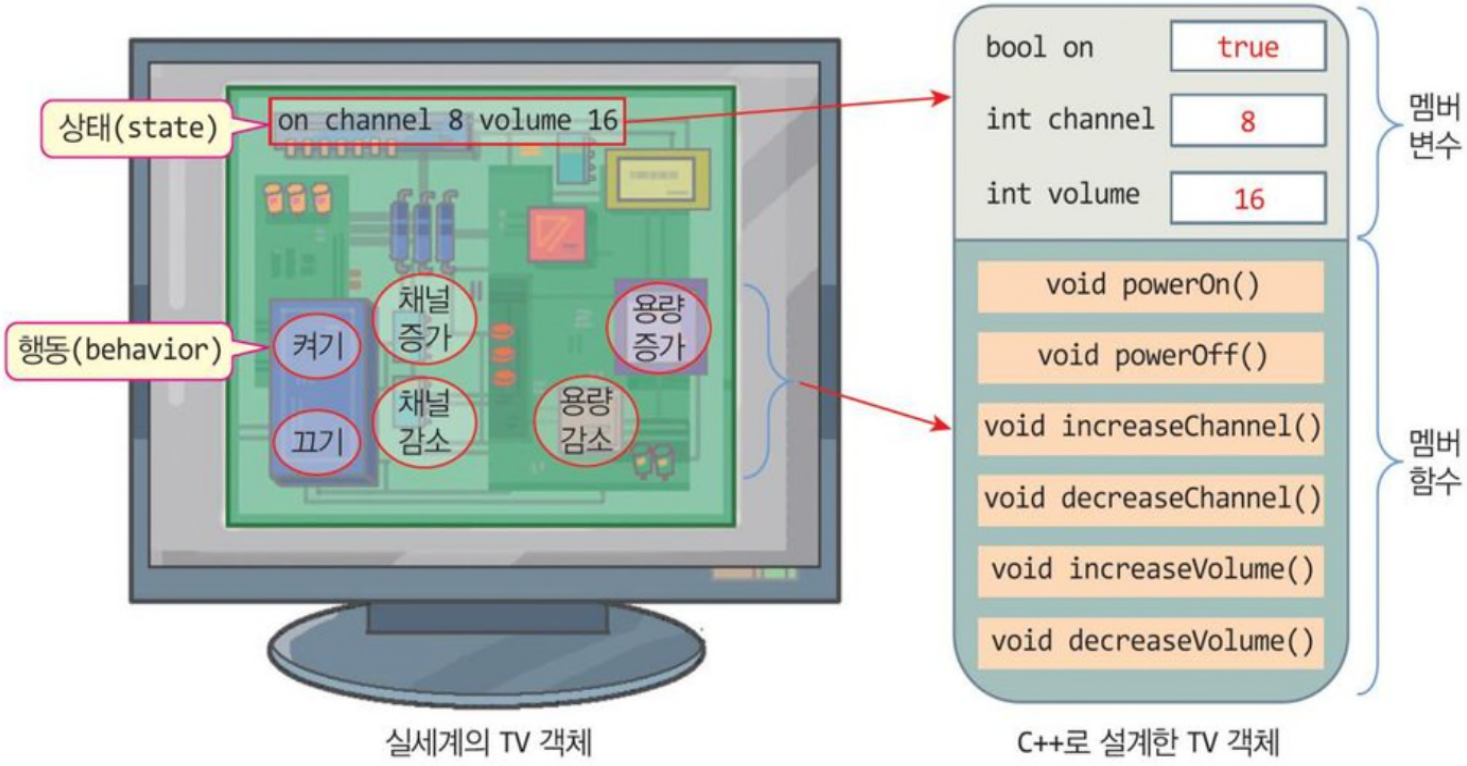
- 클래스(class): 객체를 만들어내기 위해 정의된 설계도 (i.e. 멤버변수/함수)



TV를 예시로 설명한 C++의 객체와 클래스

➢ C++: 객체지향 프로그래밍 언어

- 객체(object): 행동(behavior, method)과 상태(state, data)의 묶음

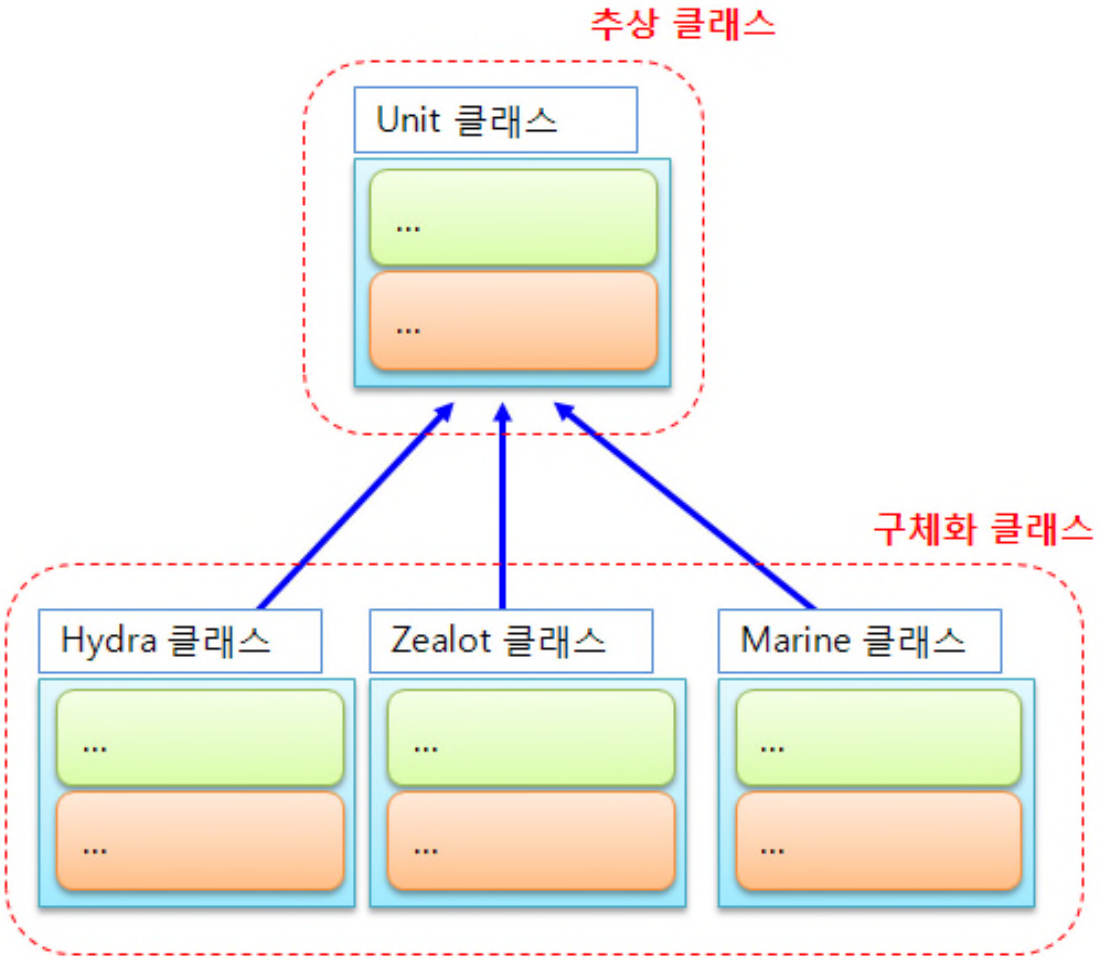


- 상태
  - on/off 속성 – 현재 작동 중인지 표시
  - 채널(channel) – 현재 방송중인 채널
  - 음량(volume) – 현재 출력되는 소리 크기
- 행동
  - 켜기(power on)
  - 끄기(power off)
  - 채널 증가(increase channel)
  - 채널 감소(decrease channel)
  - 음량 증가(increase volume)
  - 음량 줄이기(decrease volume)

TV를 예시로 설명한 C++의 객체

➢ C++: 데이터 추상화 (data abstraction)

- 불필요한 자료는 숨기고 중요한 것만 표현
  - 객체들이 공통으로 사용할 변수/함수가 무엇인지 생각하고 구현해야 하는 클래스



스타크래프트를 예시로 설명한 C++의 데이터 추상화

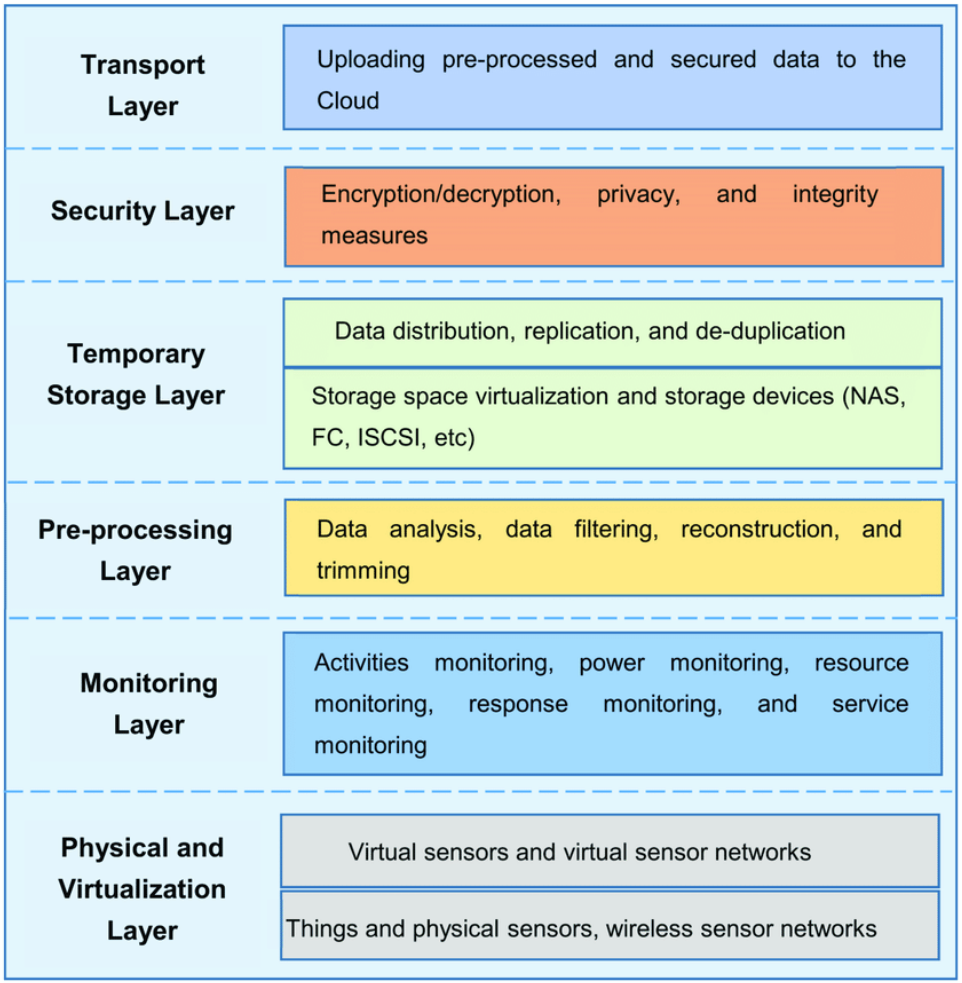
예

윈도 환경에서 가동될 메모장을 만들 예정입니다.
어차피 상속받아 쓸 건데
마우스 가동 프로그램을
세세히 알 필요가 있을까요?

추상화 클래스의 기능을 정확히 알고 있다면
구체화 단계에서는 세세히 알 필요는 없음

➢ C++: 레이어 시스템 (layered system)

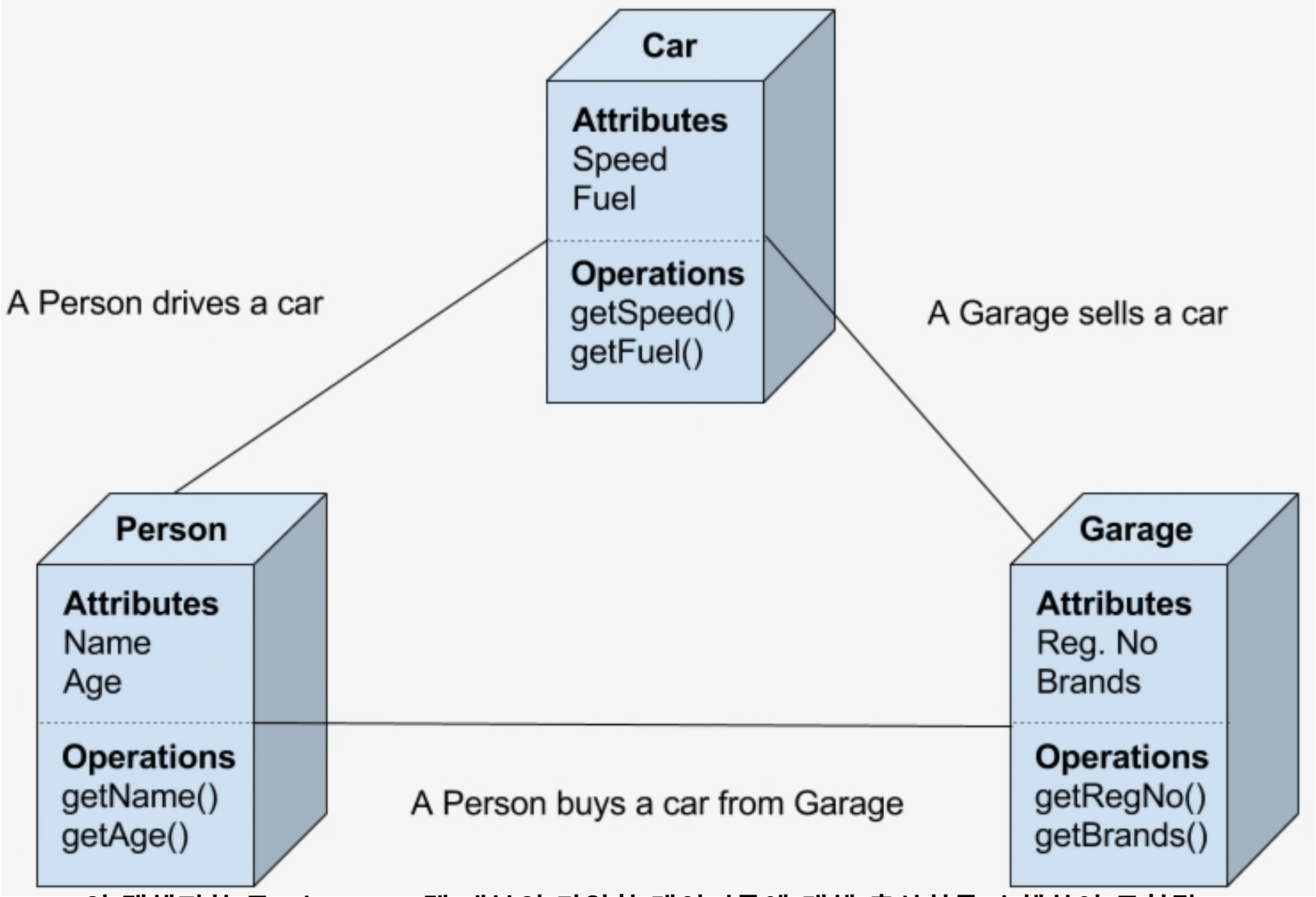
- 하위 계층은 상위 계층에게 필요한 메소드/함수 제공
  - 하위→상위: 더 높은 추상화를 제공 (i.e. 인접한 레이어들 간의 유지보수로 프로그래밍 가능)



레이어 방식의 C++ 프로그래밍 개발 예시

C vs. C++

- **C++: 객체지향 구조(object-oriented organization)**



C++의 객체지향 구조는 프로그램 내부의 다양한 데이터들에 대해 추상화를 수행하여 구현됨

C vs. C++

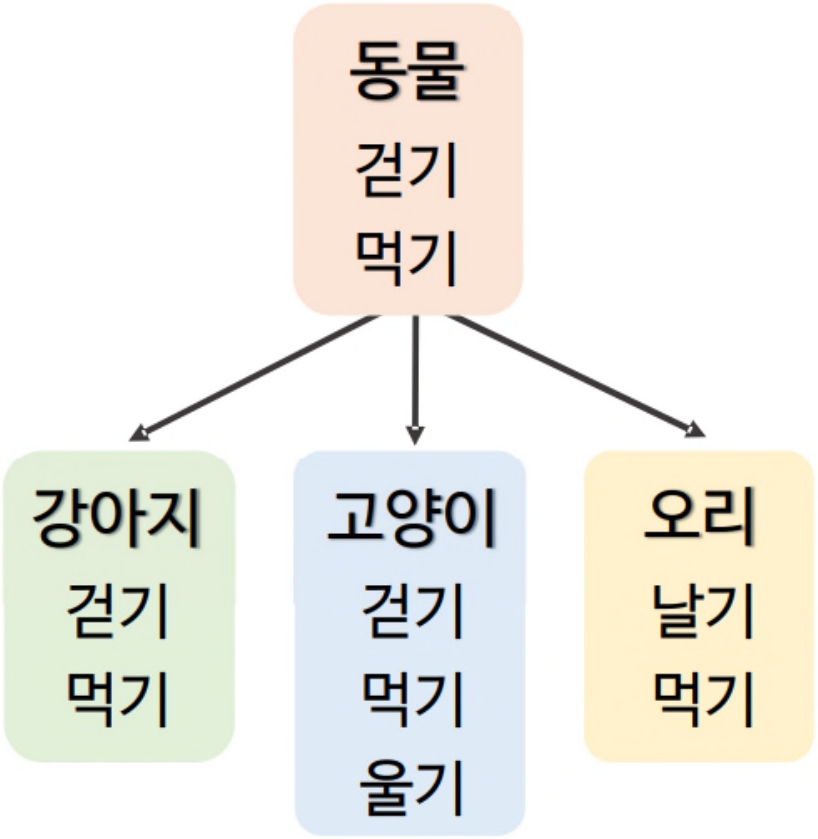
➢ 한 눈에 표로 정리하는 C와 C++

	C	C++
프로그래밍 방식	구조적 프로그래밍	객체지향 프로그래밍
프로그램 분할 방법	기능	객체
구현 단위	함수	클래스
규모	중소형 프로그램	중대형 프로그램

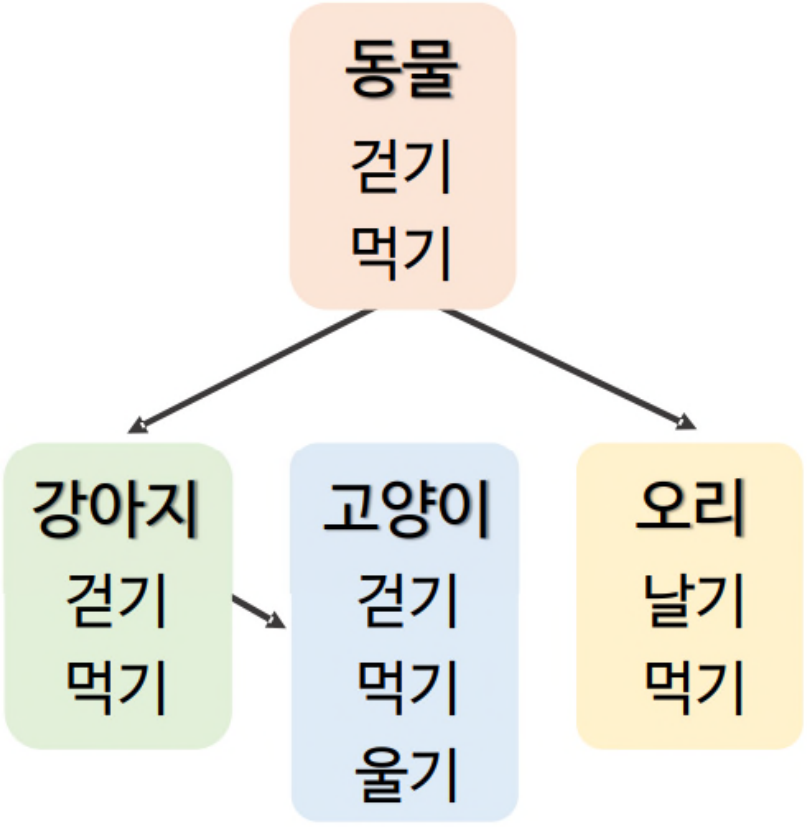
사용하려고 생각한 언어의 특성을 생각해보자

➢ C++: 객체지향 프로그래밍 언어

- 상속성(inheritance)
  - 하나의 클래스로부터 새로운 클래스를 유도할 수 있음
    - 객체단위로 상속 또는 상속 후 수정하여 재사용 (다중상속도 가능)



C++의 객체지향 프로그래밍 특성 중 상속성 개념예시



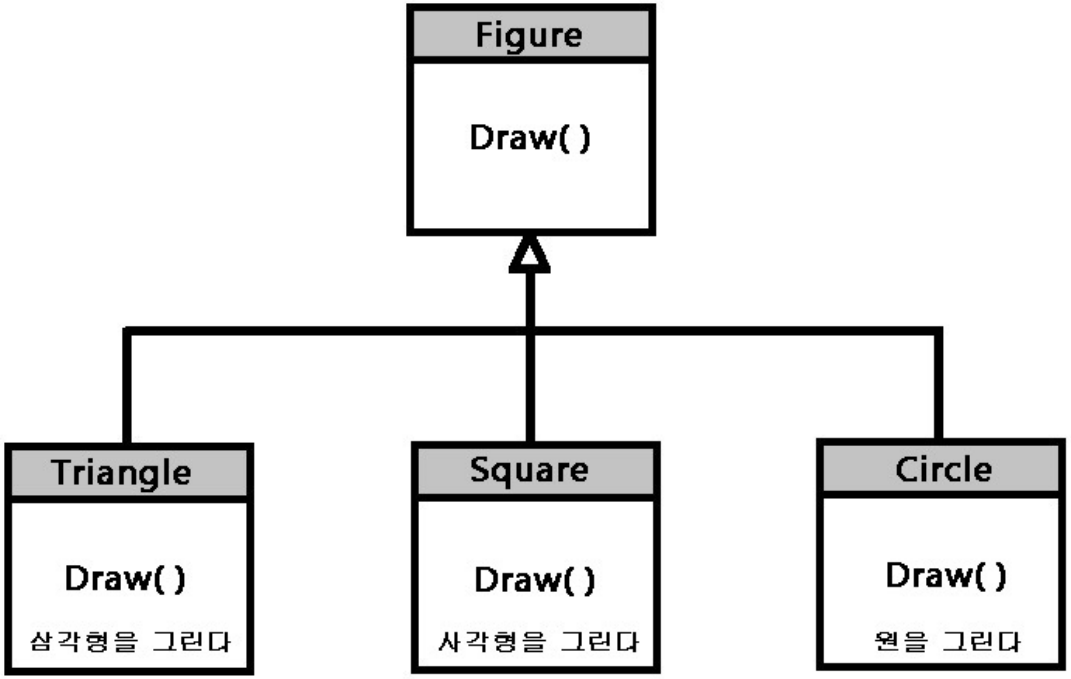
다중상속 개념 예시

➢ C++: 객체지향 프로그래밍 언어

- 다형성(polymorphism)
  - 이름이 같은 연산자와 함수를 여러번 정의 가능
  - 상황에 따라 적당한 연산자나 함수를 프로그램이 스스로 선택할 수 있음
    - 메소드 오버로딩: 같은 이름의 함수를 중복정의(e.g. 컴파일러가 자료형에 적합한 메소드 찾아서 지정)
    - 메소드 오버라이딩: 같은 이름의 함수를 재정의(e.g. 이미 정의된 메소드를 재정의)

```cpp
// 1) 오버로딩
// => 함수 중복 정의 = 함수 이름의 재사용
int Add(int a, int b) // 정수형 타입의 합을 구하는 함수
{
    int result = a + b;
    return result;
}

float Add(float a, float b) // 실수형 타입의 합을 구하는 함수
{
    float result = a + b;
    return result;
}
```

오버로딩 예시



오버라이딩 예시

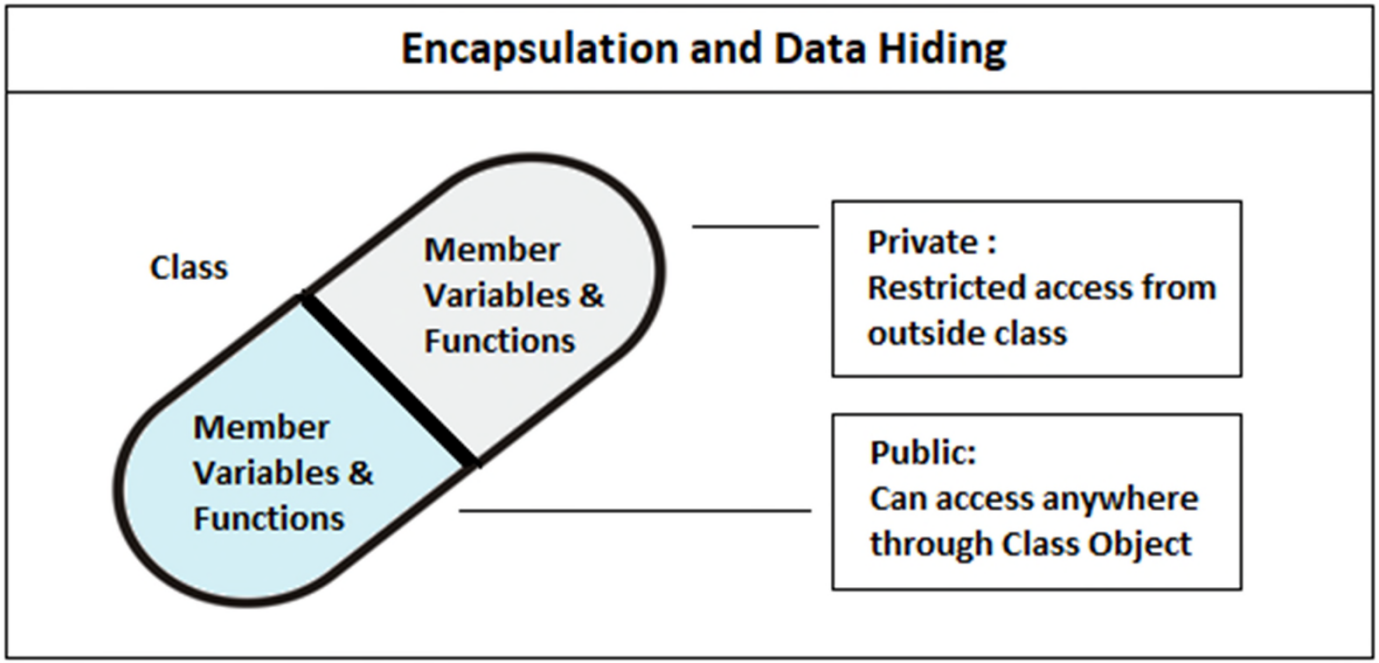
➢ C++: 객체지향 프로그래밍 언어

- 은닉화(hiding)
  - 내부 데이터/연산을 외부에서 접근하지 못하도록 처리 (i.e. 격리(isolation))
  - 변수에 접근지정자를 private 로 지정
  - set/get 측면의 함수를 정의하여 변수 접근을 제어

클래스 내부에 은닉화하여 내부 데이터/연산을 외부에 공개하지 않음

➢ C++: 객체지향 프로그래밍 언어

- 캡슐화(Encapsulation)
  - 실제로 구현되는 부분을 외부에 드러나지 않도록 감싸 이용방법만을 알려주는 설계
    - 데이터 구조와 데이터를 다루는 방법들을 결합 시켜 묶어줌
    - 객체가 맡은 역할을 수행하기 위한 하나의 목적을 한데 묶어야 함
    - 오로지 함수를 통해서만 접근할 수 있도록 설계해야 함



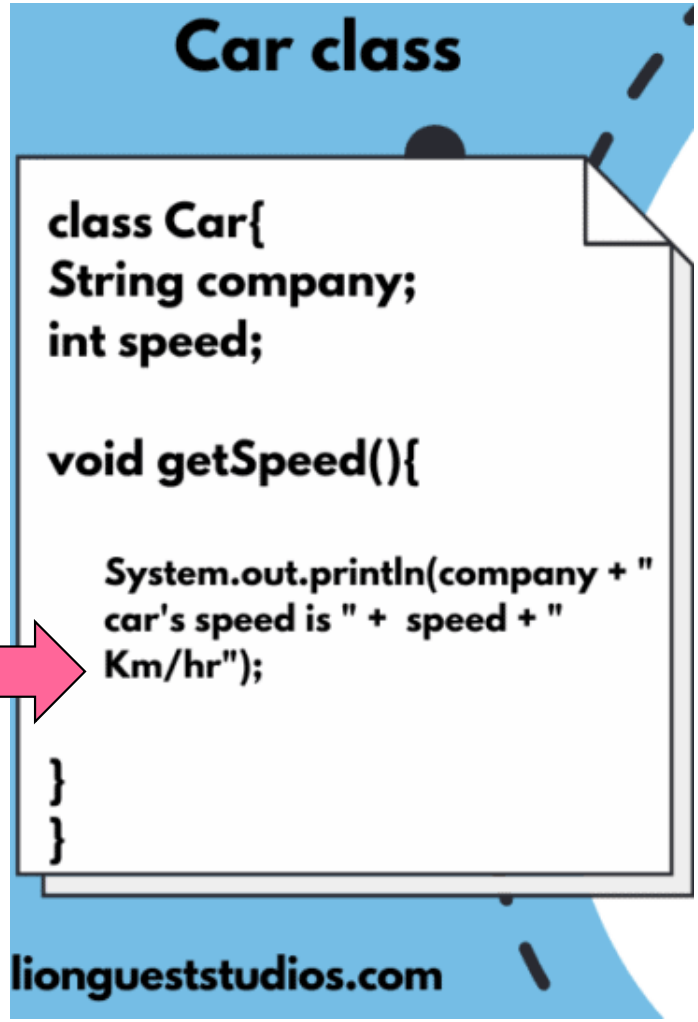
외부에서 접근 가능한 변수/함수와 내부에서만 사용하는 변수/함수에 대한 구분

C++: Class 101::클래스

➢ 사물의 상태/특성/기능을 정리하여 프로그램으로 표현하기 위해 필요한 개념

- 클래스: 멤버 변수와 멤버 함수로 구성
  - C언어의 구조체와 유사→ 접근 지정자의 차이 (C의 구조체: public, C++의 클래스: private)
  - public/private의 이해 (캡슐화)
- 멤버 변수
  - 클래스에 선언한 변수
- 멤버 함수
  - 클래스에 선언한 함수

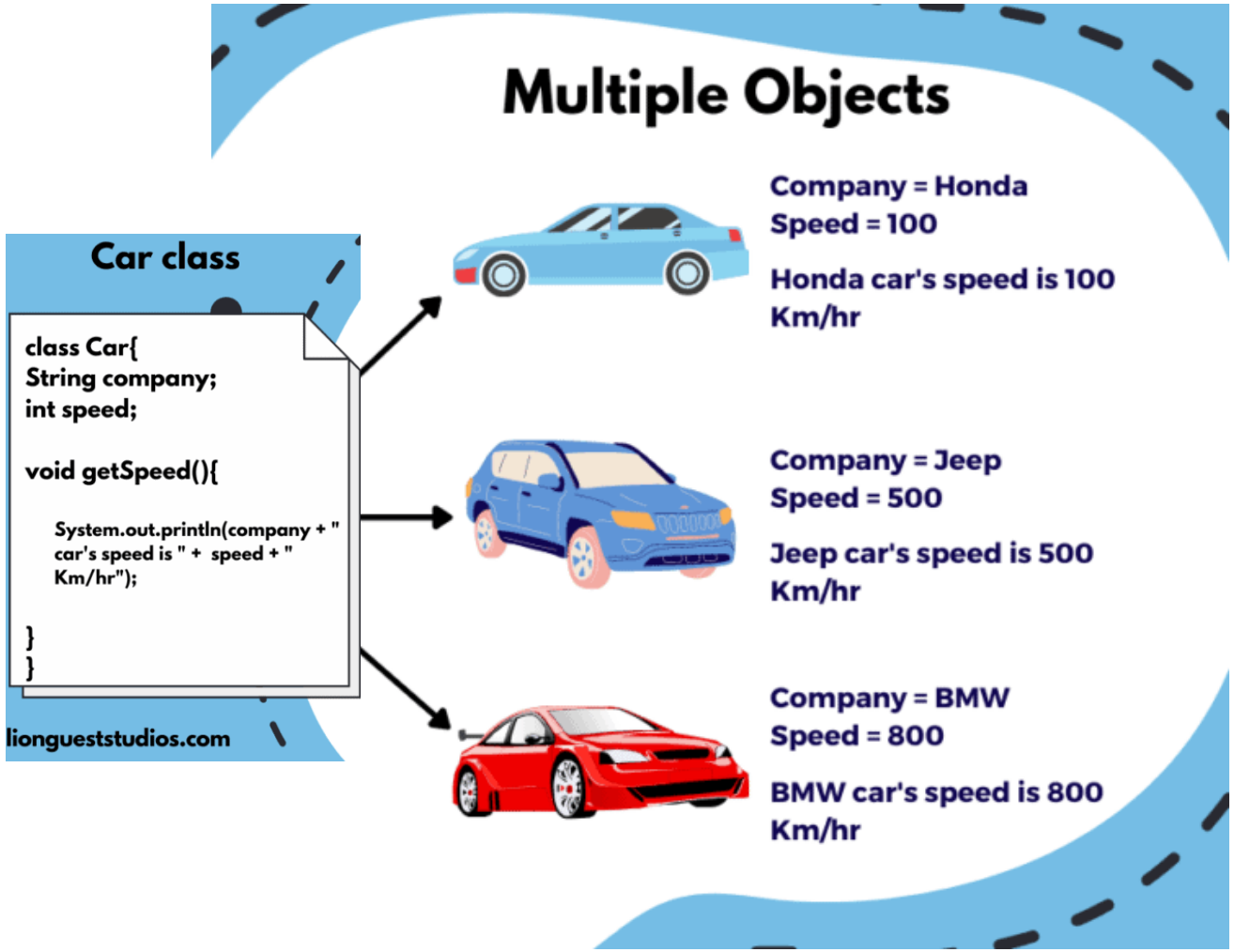
```
class 클래스 이름{
접근 제어 지시자:
    멤버변수:
    멤버함수
};
```



C++: Class 101::객체

➢ 행동(behavior, method)과 상태(state, data)의 묶음

- 클래스를 main() 또는 필요한 함수에서 사용하기 위해 선언하는 코딩
  - 객체를 선언하여 접근 가능한 멤버 변수와 멤버 함수를 사용

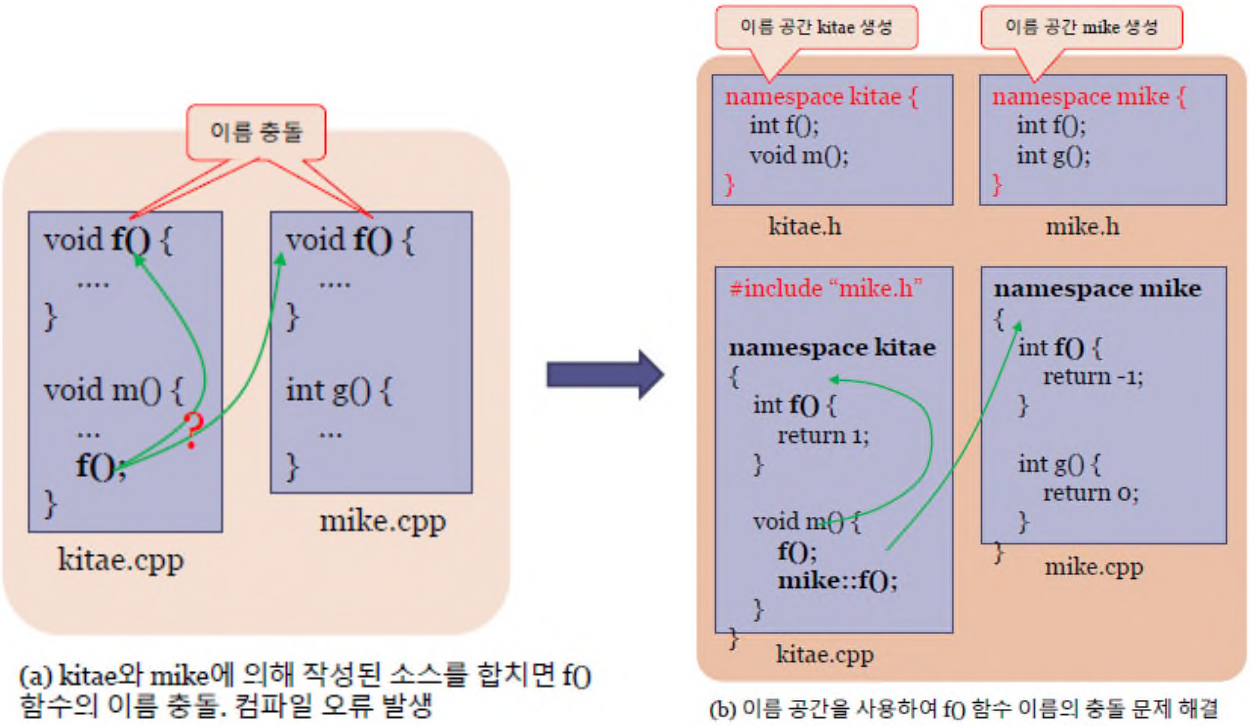


C++: Class 101::namespace

➢ 개발과정 중 함수/구조체/클래스명이 같을 때 발생하는 충돌을 방지하는 코딩

- 소속을 고유하게 결정해주는 역할 → 컴파일 시 모호성에 따른 오류가 발생하지 않음
  - 중첩해서 선언 가능
    - C++의 std 등 표준 라이브러리들을 사용할 때 적용

```
namespace twoD
{
  class CalcVector
  {
  };
}

namespace threeD
{
  class CalcVector
  {
  };
}
```



(a) kitae와 mike에 의해 작성된 소스를 합치면 f() 함수의 이름 충돌. 컴파일 오류 발생

(b) 이름 공간을 사용하여 f() 함수 이름의 충돌 문제 해결

C++: Class 101::스코프 해결 연산자 (::)

➢ namespace, 클래스 이름으로 소속을 표현할 때 사용하는 연산자 (e.g. std)

- ::를 연속으로 사용하여 소속을 확실하게 표현, 중대형 규모의 협업 시 사용
  - :를 1개만 사용하면? → 클래스를 상속할 때 사용

```C++
namespace NamespaceA{
    int x;
    class ClassA {
    public:
        int x;
    };
}

int main() {

    // A namespace name used to disambiguate
    NamespaceA::x = 1;

    // A class name used to disambiguate
    NamespaceA::ClassA a1;
    a1.x = 2;
}
```

스코프 해결 연산자 :: 의 사용

```C++
#include <iostream>

class Computer { ... };
class Notebook : public Computer {
private:
    // 배터리 남은 양.
    double battery = 100.0;
};

int main(void) {
    Notebook myNotebook;

    // 컴퓨터의 기본 메소드를 사용할 수 있다.
    myNotebook.TurnOn();
    myNotebook.Display();
    myNotebook.TurnOff();

    return 0;
}
```

클래스 상속을 위한 : 연산자

C++: Class 101::this 포인터(자기참조)

➢ 모든 멤버함수에 추가되는 숨겨진 매개변수 역할을 하는 클래스 지칭 포인터

- 객체 자기자신을 지칭하는 용도로 사용
  - 클래스 내부에서 멤버함수 코딩 시 멤버변수에 접근하기 위한 역할
  - 클래스 외부에서 멤버함수 호출 시 사용한 객체를 지칭
  - 멤버함수 내부에서 입력변수명과 멤버변수명을 구분하기 위함
  - 멤버함수 체이닝(chaining) 기법

```
class Calc
{
private:
   int m_Value = 0;

public:
   Calc& Add(int value) { m_Value += value; return *this; }
   Calc& Sub(int value) { m_Value -= value; return *this; }
   Calc& Mul(int value) { m_Value *= value; return *this; }

   int GetValue() { return m_Value; }
}
```

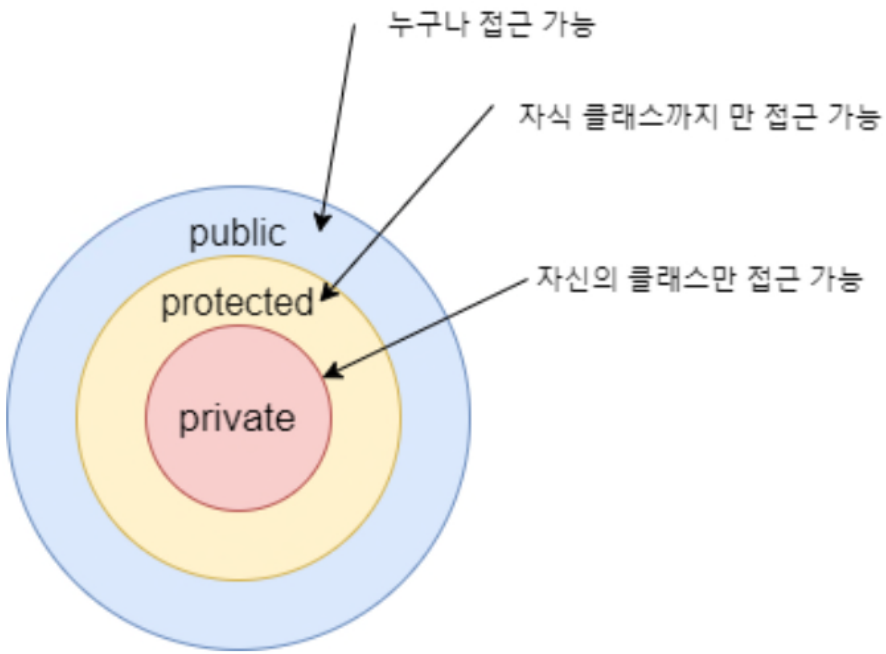
```
int main()
{
   Calc calc;

   calc.Add(5).Sub(3).Mul(4);

   cout << calc.GetValue() << endl; // 8
   return 0;
}
```

```
#include <iostream>

using namespace std;

class A {
    int num;
public:

    A(int num) {
        this->num = num;
    }

    int get_Num() {
        return num;
    }
};

int main(void) {

    A a(10);

    cout << a.get_Num() << endl;
    return 0;
}
```

C++: Class 101::데이터 멤버

➢ 클래스 내부 접근권한을 부여할 수 있음

접근 제한자	설명
private	오직 자신의 클래스안에서만 접근 가능한 멤버 변수, 메소드.
protected	자신을 상속하는 클래스까지만 접근 가능
public	어떤 클래스에서나 접근 가능



접근 범위

C++: Class 101::생성자/소멸자

➢ 클래스 생성 시 멤버함수/변수 등 클래스를 초기화하는 역할

- 클래스::클래스명(입력값(인수) 목록, 없으면 ()처리)
  - 오버로드: 입력값이 없거나, 다양하게 사용하여 초기화할 수 있음
    - 객체 배열을 생성할 때 초기화하는 역할로도 사용할 수 있음

➢ 클래스 소멸 시 멤버함수/변수 등을 처리해야 할 때 사용

- 인수값/반환값 없음
- 클래스::~클래스명()

```
class Box {
  int width;    // 데이터의 속성과 생성자, 메소드를 가진 클래스
  int height;
  int depth;
  public void Box(int w, int h, int d) {  //생성자, 클래스명과 이름이 같다
    width=w;
    height=h;
    depth=d;
  }
  public void volume() {   //메소드
    int vol;
    vol = width * height * depth;
    System.out.println("Volume is "+vol);
  }
}
```

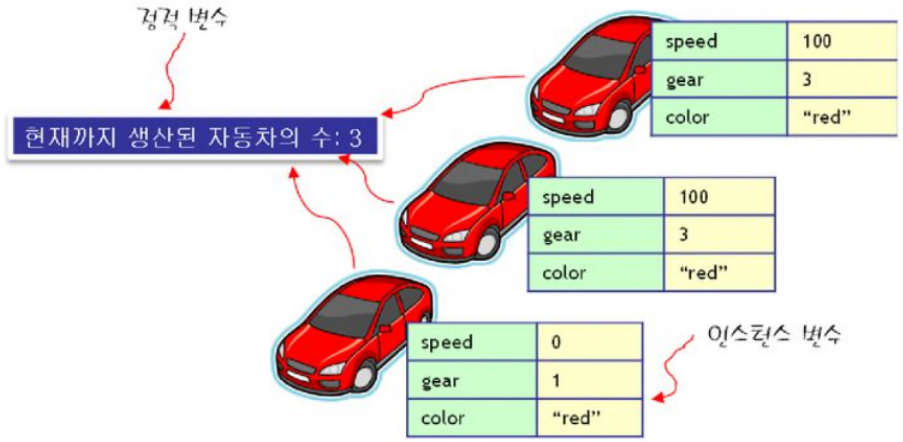
```
public class Box {
  int width;
  int height;
  int depth;
}
```

C++: Class 101::정적멤버

➢ **static으로 선언한 멤버변수/함수 → 최초 1번 초기화 후 전체 객체에서 사용**

- 클래스를 통해 선언된 객체 전체를 관리하기 위한 데이터 저장/처리용
- 정적멤버함수 내부에서는 일반 멤버변수/함수는 접근 불가

- 인스턴스 변수(instance variable): 객체마다 하나씩 있는 변수
- 정적 변수(static variable): 모든 객체를 통틀어서 하나만 있는 변수




```
class Box {
  int width;
  int height;
  int depth;
  long idNum;
  static long boxID = 100;
  static long getcurrentID(){
    return boxID++;
  }
}
```

C++: Class 101::상속

➢ 기본 클래스를 바탕으로 새로운 클래스를 만들 때, 사용하는 기능

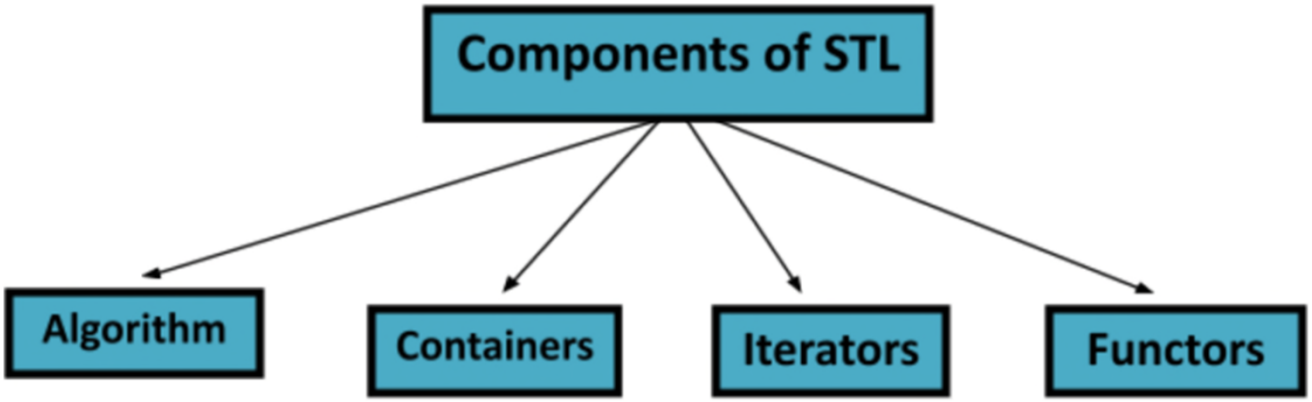
- 기본클래스의 멤버변수/함수 사용하면서 동작하는 파생클래스 작성 가능
  - 2개 이상의 기본클래스 상속 가능
- 추상화 클래스: 순수한 가상(virtual)함수를 하나 이상 가지는 클래스
  - 객체생성이 불가함 → 이를 상속하여 기본클래스를 만들고 객체생성 가능
    - 기존 코드에 살을 덧붙이듯 새로운 코드를 작성할 수 있음

클래스 상속의 이해

C++: Advanced::Standard Template Library(STL)

➢ 자료구조/알고리즘을 편하게 사용할 수 있도록 제공되는 라이브러리

- 컨테이너 + 반복자 + 알고리즘
  - 컨테이너: 데이터를 보다 다양한 방법으로 저장하도록 만들어주는 역할
    - vector, deque, map, list, etc.
  - 반복자: 컨테이너 내부 데이터를 순회하면서 특정 위치를 나타내주는 역할
    - begin(), end(), etc.
  - 알고리즘: 검색/정렬/삭제/연산 등을 편하게 수행하도록 도와주는 역할
    - sort(), etc.

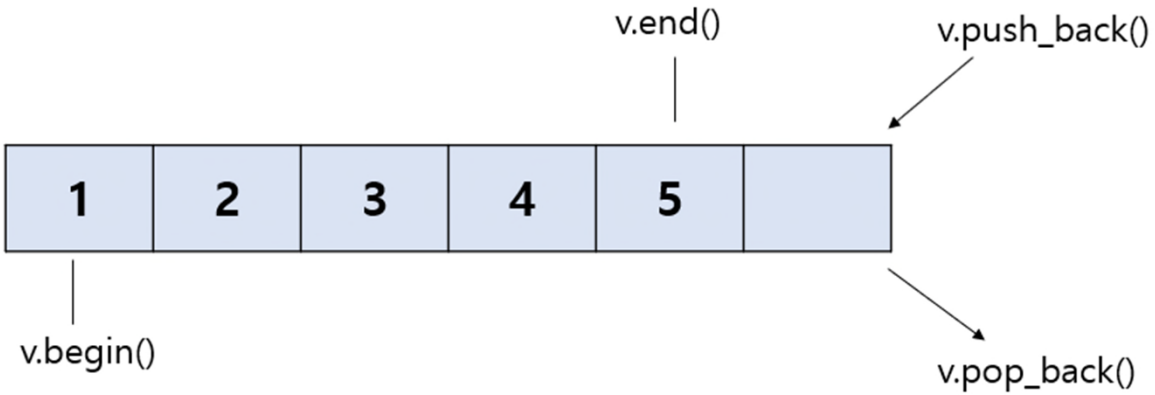


C++의 STL 구성

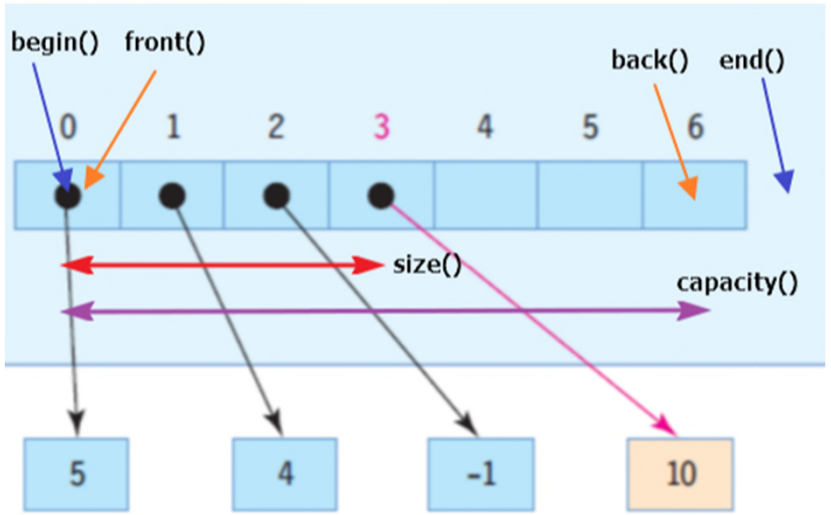
C++: Advanced::std::vector

➢ 크기를 자유롭게 변경할 수 있는 C++의 적응형 배열 (#include <vector>)

- 구조체와 함께 사용하면 수학/물리적으로 명시적인 코딩 가능
  - 단, size()연산자를 통해 현재 크기에 따른 적절한 처리 필요 (i.e. segmentation fault)



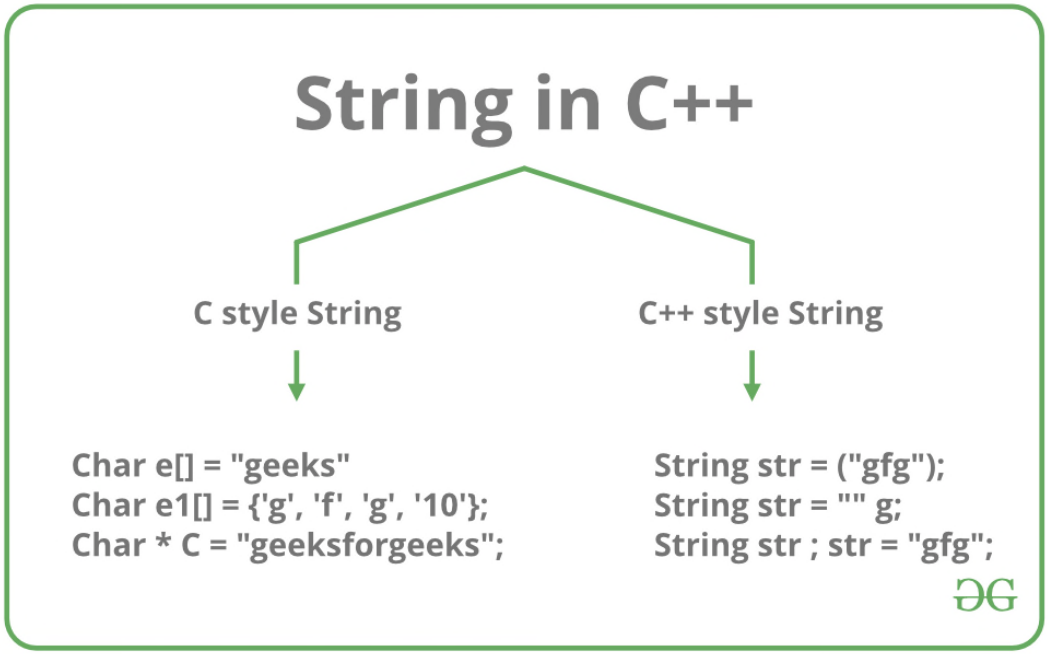
vector를 다루기 위해 이 정도는 기본으로 알아야 함!



C++: std::string

➢ 문자열을 보다 편하게 사용할 수 있도록 도와주는 역할

- #include <string>으로 사용
  - string 변수명("(문자열)") 또는 string 변수명; 후 변수명 = "문자열"; 로 선언
    - printf 출력 시 변수명.c_str()을 수행해줘야 함
    - 덧셈연산 가능, front(), back()으로 맨앞/맨뒤 인자 반환, size()/length()로 크기 반환
    - substr(시작,종료)로 해당구간에 대한 문자열만 추출
    - rfind('기호')로 해당 기호의 위치값 반환 가능

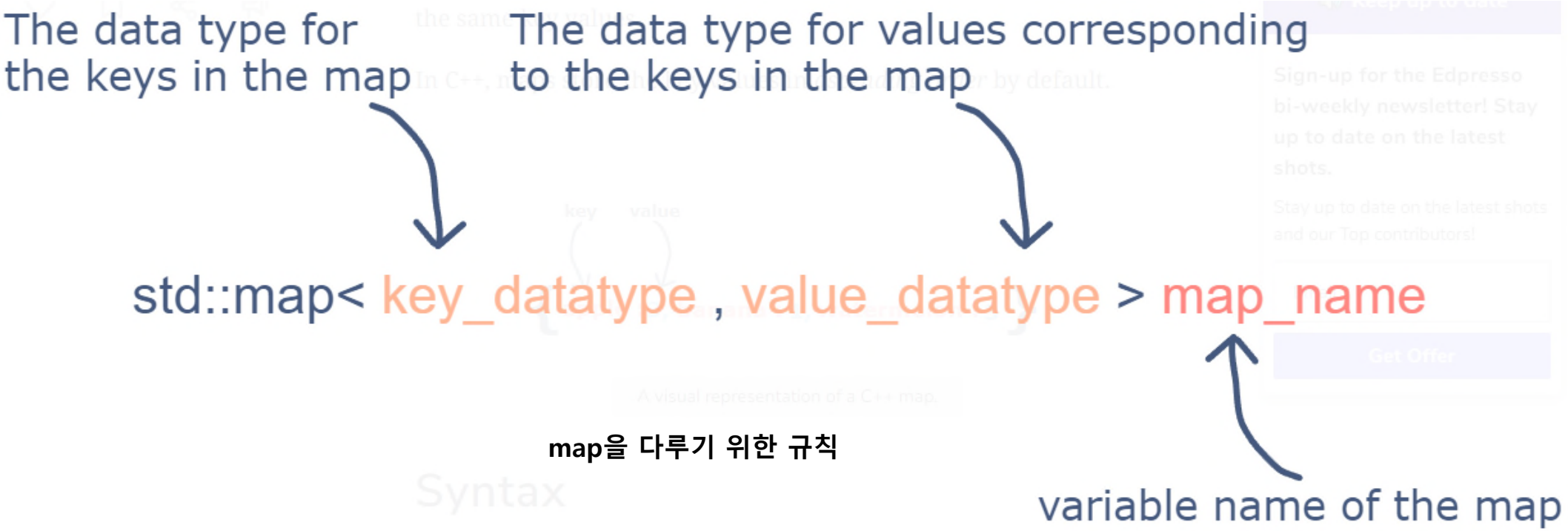


C언어와 C++의 문자열 다루기

C++: Advanced::std::map

➢ 파이썬의 dictionary와 같음, key-value 기반의 컨테이너 (#include <map>)

- 배열은 [숫자]로 값을 기억 → map은 ["문자"]로 값을 기억 (i.e. 명시적인 코딩)
  - switch-case 문에 사용 가능

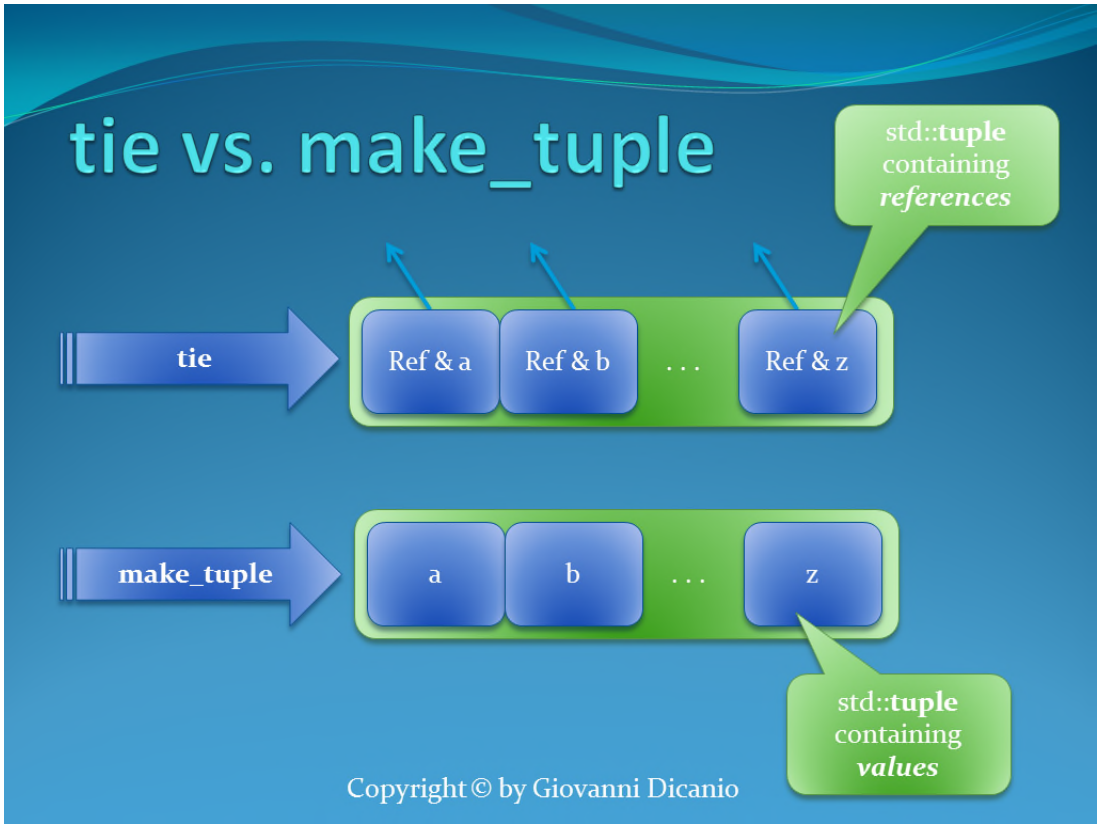


map을 다루기 위한 규칙

C++: Advanced::std::tie and tuple

➢ **함수의 출력값이 여러개일 때, 대응 가능 (#include <tuple>)**

- C언어는 structure 사용 → 명시적이지 않은 것들끼리 묶일 수 있음
- C++에서는 함수 출력시에만 잠시 사용할 수 있음

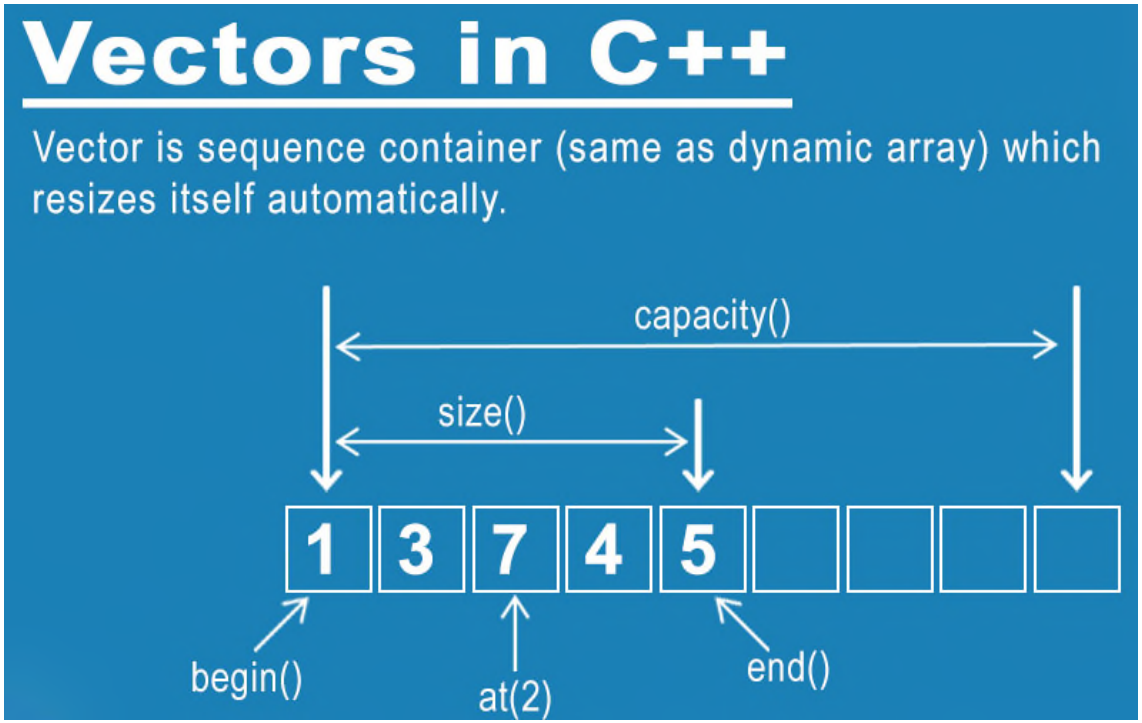


make_tuple + tie를 사용한 임시로 함수의 출력값 2개 이상 만들기

C++: Advanced::std::sort

➢ 편하게 사용 가능한 정렬 알고리즘 (#include <algorithm>)

- 단순 배열/vector에서의 정렬 및 복잡한 vector/structure 상의 정렬도 가능
  - 임의의 원소를 기준으로 전체 데이터를 정렬할 수 있음
  - 임의의 조건식을 사용할 수 있음



sort 알고리즘 사용하기

참고문헌

➢ 이론

- 두들낙서의 C/C++ 한꺼번에 배우기 (ISBN 979-11-5839-258-1), 한빛미디어, 2021

➢ 그림

- 뇌를자극하는C++ 프로그래밍, 이현창, 한빛미디어, 2011
- 열혈C++ 프로그래밍(개정판), 윤성우, 오렌지미디어, 2012
- 객체지향원리로이해하는ABSOLUTE C++ , 최영근외4명, 교보문고, 2013
- C++ ESPRESSO, 천인국, 인피니티북스, 2011
- 명품C++ Programming, 황기태, 생능출판사, 2018
- 어서와C++는처음이지, 천인국, 인피니티북스, 2018