Chapter 00. C++ 배경 및 기초

박 종 혁 교수

UCS Lab

Tel: 970-6702

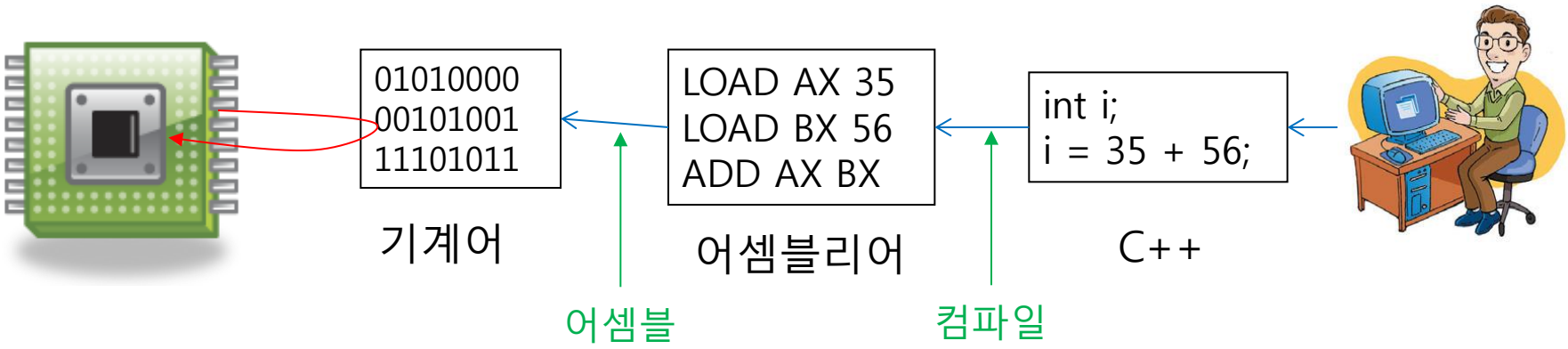
Email: jhpark1@seoultech.ac.kr

프로그래밍과 프로그래밍 언어

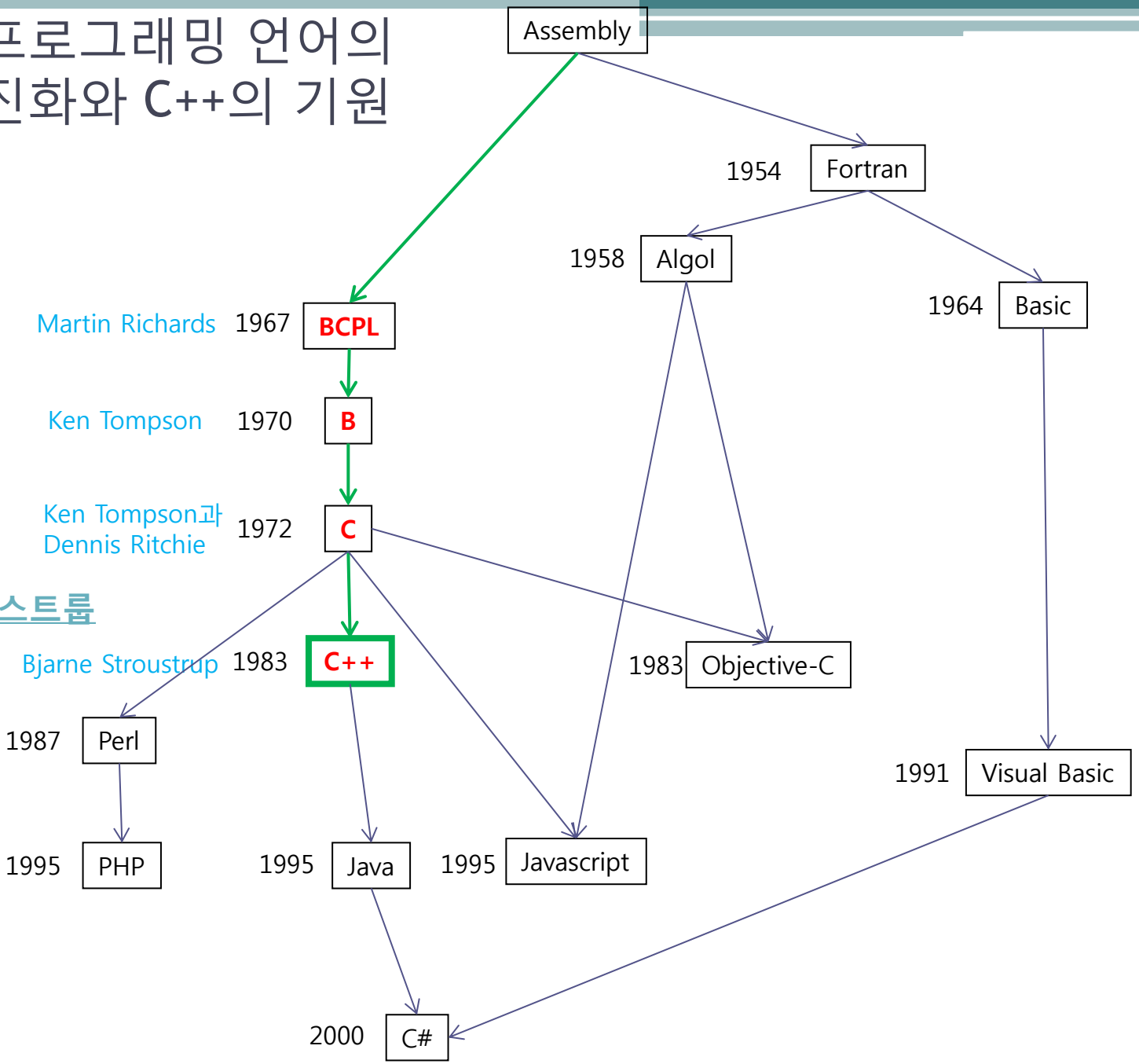
- 프로그래밍 언어
  - 기계어(machine language)
    - 0, 1의 이진수로 구성된 언어
    - 컴퓨터의 CPU는 본질적으로 기계어만 처리 가능
  - 어셈블리어
    - 기계어의 명령을 ADD, SUB, MOVE 등과 같은 상징적인 니모닉 기호(mnemonic symbol)로 일대일 대응시킨 언어
    - 어셈블러 : 어셈블리어 프로그램을 기계어 코드로 변환
  - 고급언어
    - 사람이 이해하기 쉽고 복잡한 작업, 자료 구조,알고리즘을 표현하기 위해 고안된 언어
    - Pascal, Basic, C/C++, Java, C#
    - 컴파일러 : 고급 언어로 작성된 프로그램을 기계어 코드로 변환

사람과 컴퓨터, 기계어와 고급 언어

35 + 56 = ?



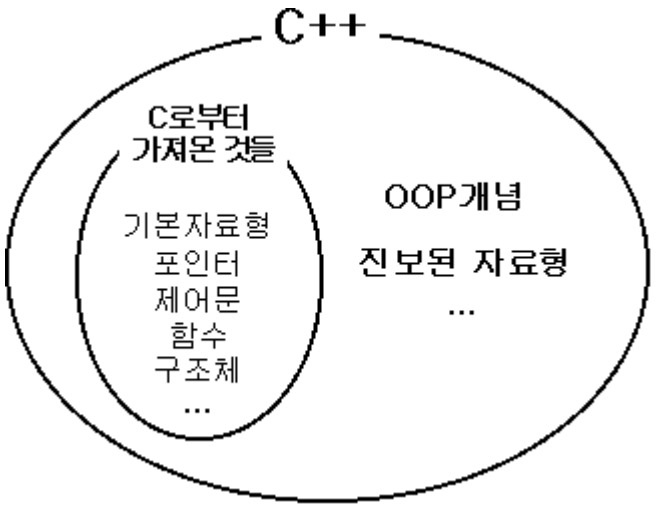
프로그래밍 언어의 진화와 C++의 기원



비야네 스트롭스트룹

C 언어로부터의 발전

- 절차지향
  (Procedural Oriented Programming)
  ↓
  객체지향
  (Object Oriented Programming)



C++ 언어의 개발

- C++는 1980년대 초에 AT&T 벨연구소의 Bjarne Stroustrup(덴마크)에 의하여 개발
- C++는 C언어를 유지, 확장한 것
- C with Classes -> C++
- C++는 C언어에 클래스 개념을 추가하고 이어서 가상 함수, 연산자 중복 정의, 다중 상속, 템플릿, 예외 처리 등이 기능이 차례로 추가

C++의 설계 철학

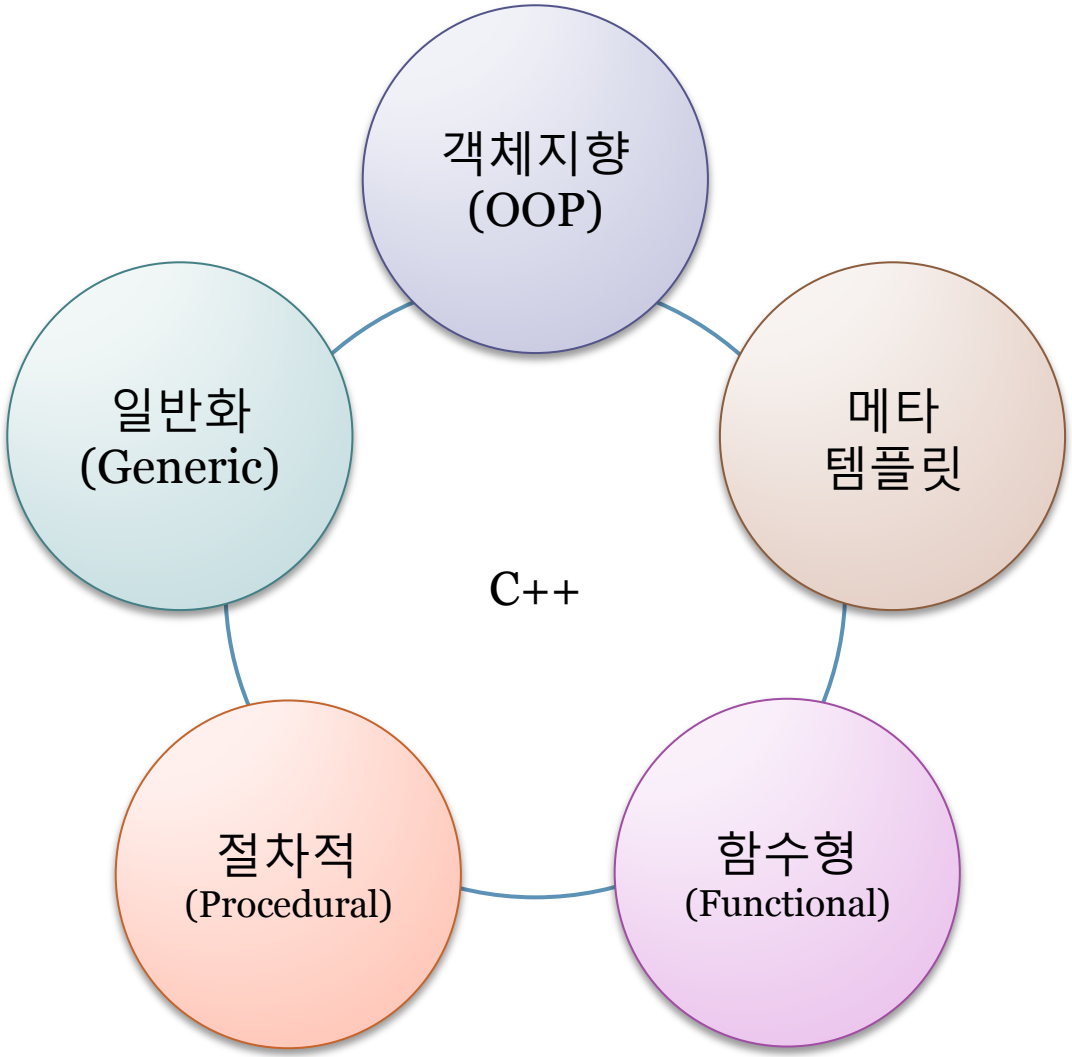
- 엄격한 타입 검사, 범용 언어, 효율적, 이식성
- 여러 가지의 프로그래밍 스타일을 지원 (절차 지향 프로그래밍, 데이터 추상화, 객체 지향 프로그래밍, 일반화 프로그래밍)
- 프로그래머가 자유롭게 선택할 수 있도록 설계
- 최대한 C와 호환
- 플랫폼에 의존적이거나 일반적이지 않은 특징은 제거

표준 C++ 프로그램의 중요성

- C++ 언어의 표준
  - 1998년 미국 표준원(ANSI, American National Standards Institute)
    - C++ 언어에 대한 표준 설정
  - ISO/IEC 14882 문서에 작성됨. 유료 문서
  - 표준의 진화
    - 1998년(C++98), 2003년(C++03), 2007년(C++TR1), 2011년(C++11)
- 표준의 중요성
  - 표준에 의해 작성된 C++ 프로그램
    - 모든 플랫폼. 모든 표준 C++ 컴파일러에 의해 컴파일
    - 동일한 실행 결과 보장
    - 운영체제와 컴파일러의 종류에 관계없는 높은 호환성
- 비 표준 C++ 프로그램
  - **Visual C++,** Borland C++ 등 컴파일러 회사 고유의 비 표준 구문
    - 특정 C++ 컴파일러에서만 컴파일
  - 호환성 결여

Multi-Paradigm C++

- 다양한 프로그래밍 패러다임 지원으로 모든 문제를 효과적으로 해결.
- 일반화, 객체지향, 메타 템플릿 프로그래밍을 이용한 코드의 쉬운 재활용.
- C언어의 절차적 특징과 강력해진 함수형 디자인.



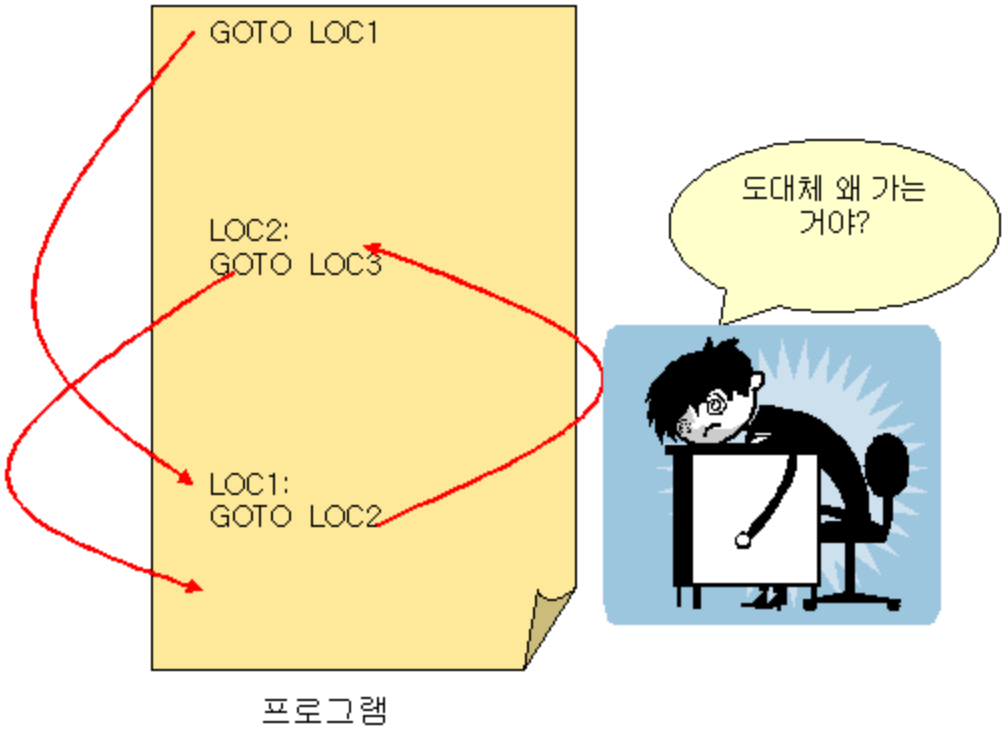
절차적 프로그래밍

- 절차적 프로그래밍에서는 데이터보다는 알고리즘(절차)을 중시

절차적 프로그래밍

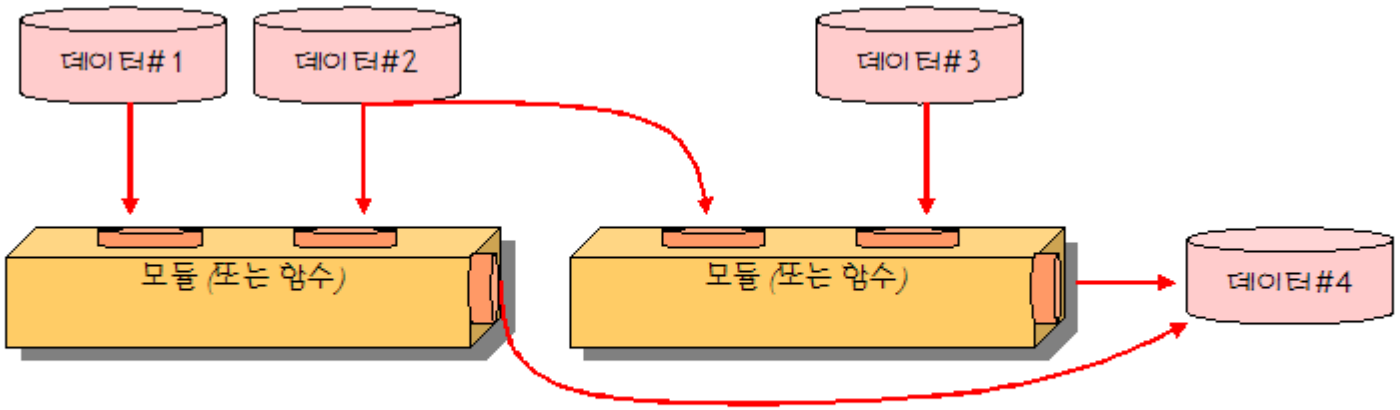
절차적 프로그래밍의 단점

- 무조건적인 점프 문장->난해



구조화 프로그래밍

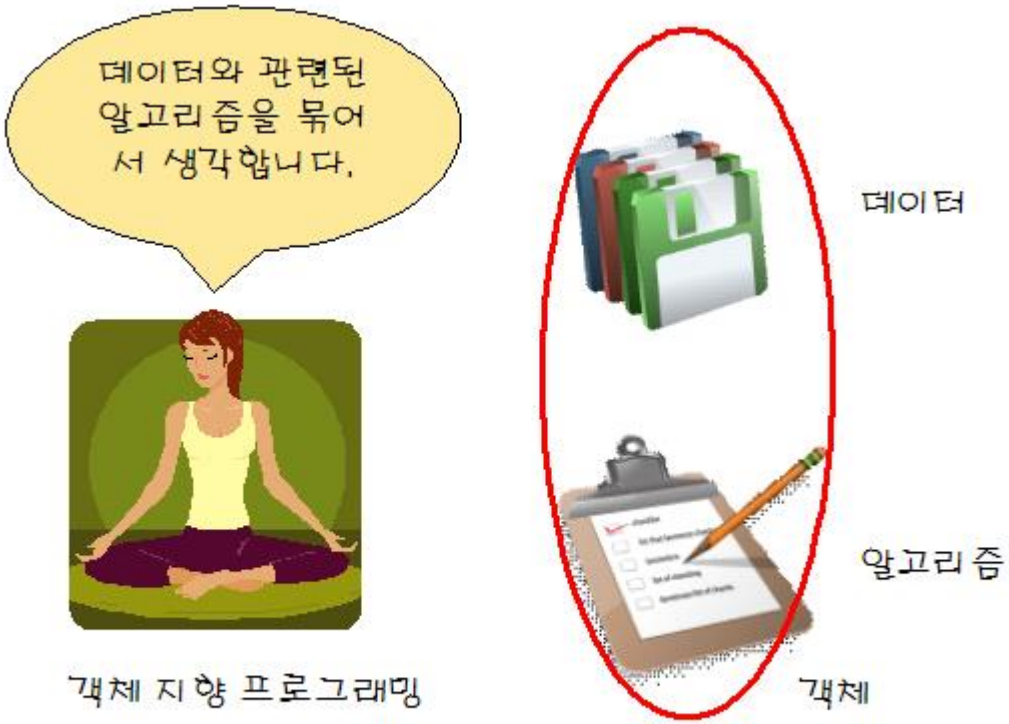
- 무조건적인 점프 없음.
- 여전히 데이터와 알고리즘은 분리되어 있음.



구조화 프로그래밍에서도 데이터와 알고리즘은 분리되어 있다.

객체 지향 프로그래밍

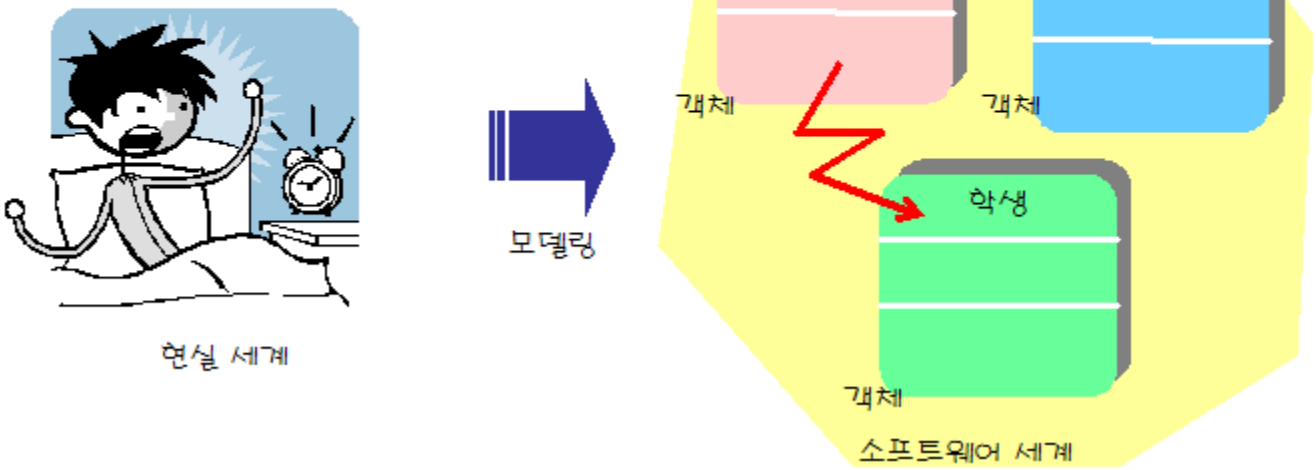
- 객체 지향 프로그래밍에서는 데이터와 알고리즘이 묶여있음.



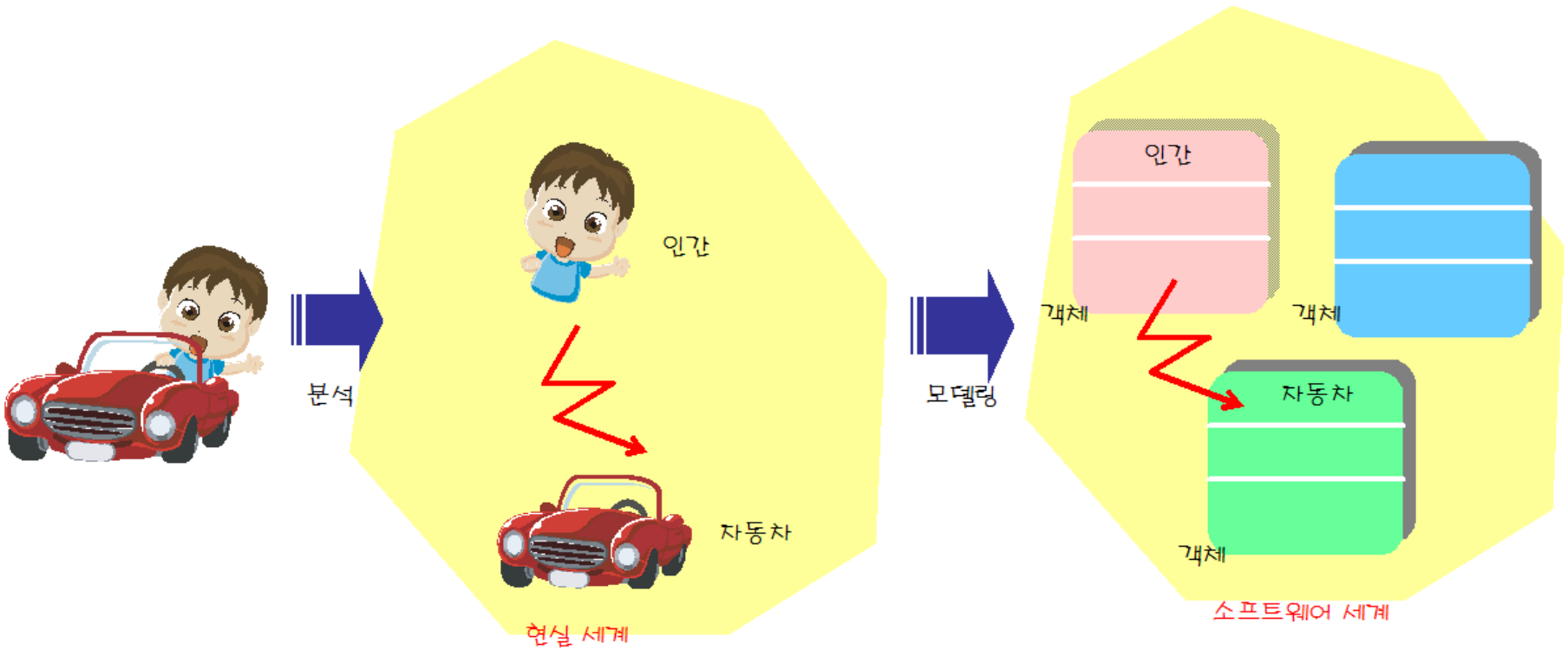
객체 지향 프로그래밍

객체 지향이란?

- 실제 세계를 모델링하여 소프트웨어를 개발하는 방법.



객체 지향의 과정

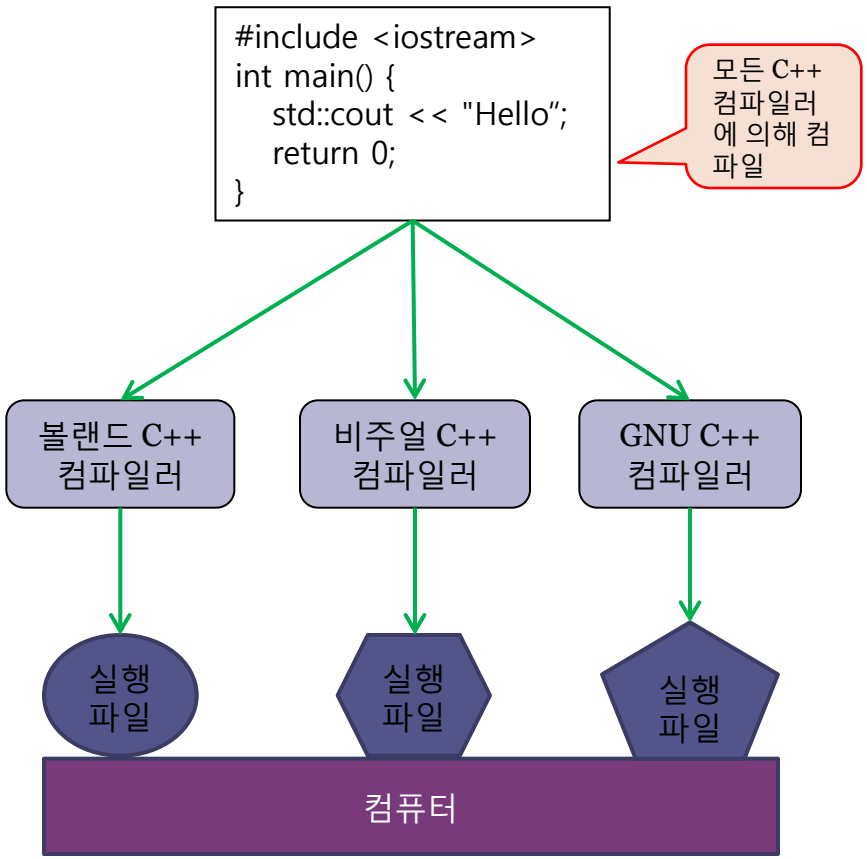


객체 지향의 개념들

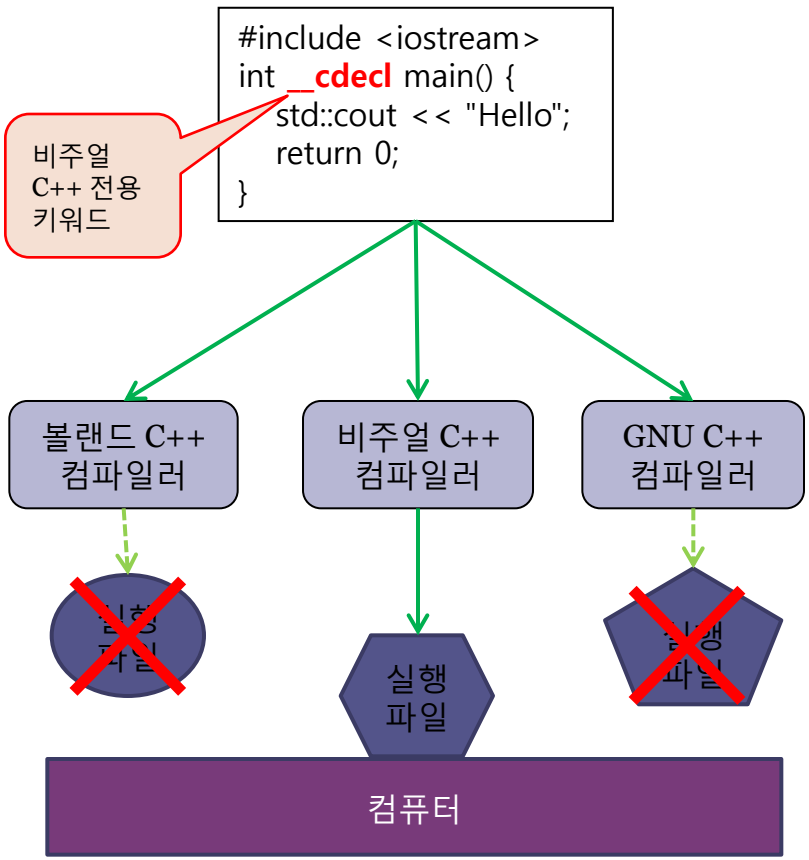
- 캡슐화(encapsulation)
- 정보 은닉(information-hiding)
- 상속(inheritance)
- 다형성(polymorphism)

표준/비표준 C++ 프로그램의 비교

표준 C++ 규칙에 따라
작성된 C++ 프로그램



표준 C++ 규칙에 따라
작성되지 않는 비주얼 C++ 프로그램



"C declarator"

C++ 언어의 주요 **설계 목적**

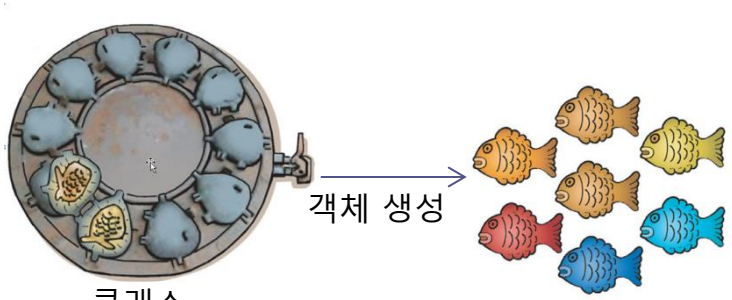
- C 언어와의 호환성
  - C 언어의 문법 체계 계승
    - 소스 레벨 호환성 - 기존에 작성된 C 프로그램을 그대로 가져다 사용
    - 링크 레벨 호환성 – C 목적 파일과 라이브러리를 C++ 프로그램에서 링크
- 객체 지향 개념 도입
  - 캡슐화, 상속, 다형성
  - 소프트웨어의 **재사용**을 통해 생산성 향상
  - 복잡하고 큰 규모의 소프트웨어의 작성, 관리, 유지보수 용이
- 엄격한 타입 체크
  - 실행 시간 오류의 가능성을 줄임
  - 디버깅 편리
- 실행 시간의 효율성 저하 최소화
  - 실행 시간을 저하시키는 요소와 해결
    - 작은 크기의 멤버 함수 잦은 호출 가능성 -> 인라인 함수로 실행 시간 저하 해소

C 언어에 추가한 기능

- 인라인 함수(inline function)
  - 함수 호출 대신 함수 코드의 확장 삽입
- 함수 중복(function overloading)
  - 매개 변수의 개수나 타입이 다른 동일한 이름의 함수들 선언
- 디폴트 매개 변수(default parameter)
  - 매개 변수에 디폴트 값이 전달되도록 함수 선언
- 참조와 참조 변수(reference)
  - 하나의 변수에 별명을 사용하는 참조 변수 도입
- 참조에 의한 호출(call-by-reference)
  - 함수 호출 시 참조 전달
- new/delete 연산자
  - 동적 메모리 할당/해제를 위해 new와 delete 연산자 도입
- 연산자 재정의(operator overloading)
  - 기존 C++ 연산자에 새로운 연산 정의
- 제네릭 함수와 클래스
  - 데이터 타입에 의존하지 않고 일반화시킨 함수나 클래스 작성 가능

C++ 객체 지향 특성 - 캡슐화

- 캡슐화(Encapsulation)
  - 데이터를 캡슐로 싸서 외부의 접근으로부터 보호
  - C++에서 클래스(class 키워드)로 캡슐 표현
- 클래스와 객체
  - 클래스 - 객체를 만드는 틀
  - 객체 - 클래스라는 틀에서 생겨난 실체
  - 객체(object), 실체(instance)는 같은 뜻



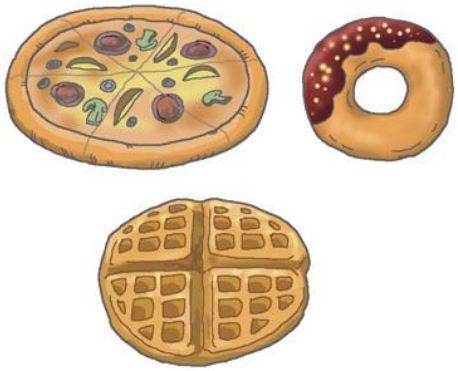
클래스
(객체를 정의하는 틀)

객체들 - 실체

멤버들

```
class Circle {
  private:
    int radius; // 반지름 값
  public:
    Circle(int r) { radius = r; }
    double getArea() { return 3.14*radius*radius; }
};
```

원을 추상화한 Circle 클래스

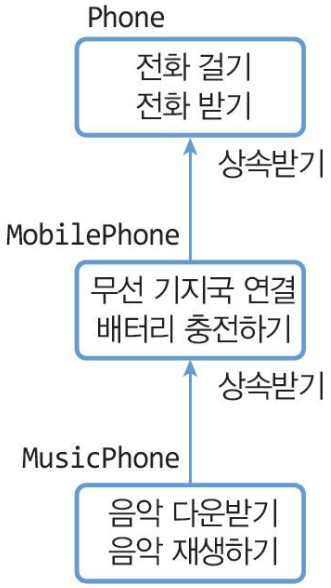


원 객체들(실체)

C++ 객체 지향 특성 - 상속

- 객체 지향 상속(Inheritance)
  - 자식이 부모의 유전자를 물려 받는 것과 유사

- C++ 상속
  - 객체가 자식 클래스의 멤버와 부모 클래스에 선언된 모양 그대로 멤버들을 가지고 탄생



상속 관계 표현

```
class Phone {
   void call();
   void receive();
};

class MobilePhone : public Phone {
   void connectWireless();
   void recharge();
};

class MusicPhone : public MobilePhone {
   void downloadMusic();
   void play();
};
```

Phone을 상속받는다.

MobilePhone을 상속받는다.

C++로 상속 선언

전화기

휴대 전화기

음악 기능 전화기

C++ 객체 지향 특성 - 다형성

- 다형성(Polymorphism) 多形性
  - 하나의 기능이 경우에 따라 다르게 보이거나 다르게 작동하는 현상
  - 연산자 중복, 함수 중복, 함수 재정의(overriding)

```
2 + 3                          --> 5
"남자" + "여자"                --> "남자여자"
redColor 객체 + blueColor 객체  --> purpleColor 객체
```

\+ 연산자 중복

```
void add(int a, int b) { ... }
void add(int a, int b, int c) { ... }
void add(int a, double d) { ... }
```

add 함수 중복

함수 재정의(오버라이딩)

정리) C와 C++

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

참고문헌

- C++ ESPRESSO, 천인국 저, 인피니티북스, 2011
- 김용성의 C&C++ 완벽가이드 (2nd Edition), 영진출판사, 2011
- 명품 C++ Programming, 황기태 , 생능출판사, 2013

Q&A