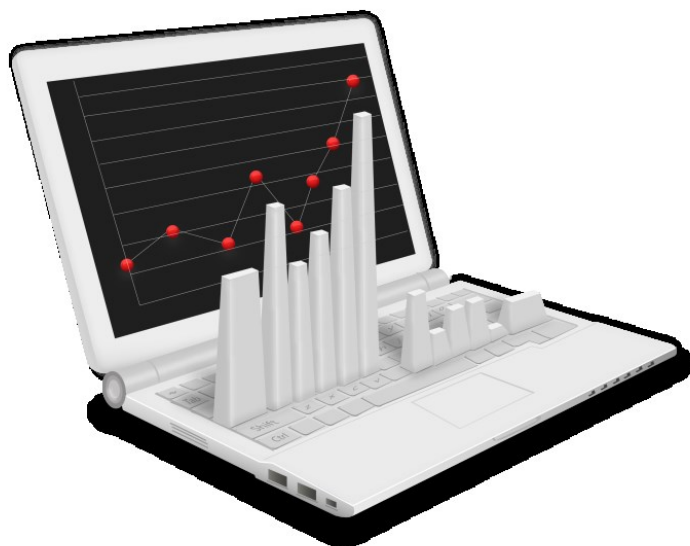


프로그래밍 언어론

함수 설계 시 고려사항
(Design Issues of Functions)

컴퓨터공학과
이만호



함수 설계 시 고려 사항

학습 목표

함수 설계 시 고려해야 할 사항들과 사용자 정의 Overloaded Operator(중복 연산자) 및 Coroutine 등에 대해서 학습한다.

학습 내용

- 함수 설계 시 고려 사항
- 사용자 정의 Overloaded Operator
- Coroutine



목 차

- 들어가기
- 학습하기
 - 함수 설계 시 고려 사항
 - 사용자 정의 Overloaded Operator
 - Coroutines
- 평가하기
- 정리하기



알고가기 1



다음 항목 중에서 C 언어의 함수(function)가 return할 수 있는 값의 type이 아닌 것은?

- a. int
- b. 배열
- c. struct
- d. pointer

확인



알고가기 2



C 언어에서 사용할 수 있는 연산자(operator) 중에서 overloaded operator(중복 연산자)가 아닌 것은?

- a. +
- b. /
- c. %
- d. &&

확인



| 함수 설계 시 고려사항

- ➔ Parameter로 인한 side-effect 발생을 허용할 것인가?
- ➔ 어떤 type의 값이 return될 수 있는가?
- ➔ 몇 개의 값을 return할 수 있는가?
- ➔ 사용자 정의 overloaded operator(중복 연산자)를 허용할 것인가?

※ First-class Object(객체)

▶ 특성

- 변수에 저장할 수 있다.
- Subprogram에게 parameter로 전달할 수 있다.
- Function의 결과로 return이 가능하다.

▶ 함수(function)가 first-class object인가?

- 대부분의 명령형 언어에서 first-class object가 아님
 - ➔ JavaScript, Python, Ruby, Lua에서는 first-class object임
- 대부분의 함수형 언어에서 first-class object임



| 수학적 함수 vs. 언어의 함수

수학적 함수 : $z = f(x, y) = \dots$

→ 함수를 계산하려면 **독립변수**(x,y)의 값이 주어져야 한다.

- ▶ $a = f(4, 7)$
- ▶ 주어진 **독립변수**의 값을 이용하여 계산해서 결과 값 하나를 얻는다.
- ▶ 함수 값 계산 과정에서 어떠한 다른 것의 값을 변화시키지 **않는다**.

→ 수학적 함수는 **참조 투명성**(referential transparency)이 특징이다.

- ▶ 함수의 값은 독립변수에만 **의존한다**.
- ▶ 동일한 독립변수 값에 대해서, 함수 값 계산 결과는 항상 동일하다.

언어의 함수 : `double fun(double x, double y) { ... }`

→ 수학적 함수를 modeling한 것임

→ 함수를 계산하려면 **parameter**(x,y)로 값이 전달되어야 한다.

- ▶ $a = \text{fun}(4.0, 7.0)$
- ▶ 전달된 **parameter**의 값을 이용하여 실행한 결과로 값 하나를 생성하여 return한다.
- ▶ 함수 실행 과정에서 parameter나 비지역변수 값을 변화시키지 **않아야 한다**.

→ 함수는 **side-effect**를 발생시키지 **않아야 한다**.

- ▶ 함수의 값은 parameter에만 **의존해야 한다**.
- ▶ 동일한 parameter 값에 대해서, 함수는 항상 동일한 값을 **return해야 한다**.



| 함수의 Side-effect

함수의 Side-effect(부수효과)

- ➔ 함수를 실행한 후, 실행환경에 변화가 발생
 - ▶ 호출자(caller)의 지역변수 값의 변화
Parameter
 - ▶ 비지역변수 값의 변화
Static ancestor들의 지역변수 (전역변수 포함)
 - ▶ 함수 실행 과정에서 입출력 실행

```
foo(...) { ... }
...
call foo(...);
...
```

Parameter로 인한 Side-effect

- ➔ Side-effect를 허용
 - > Inout mode의 parameter 전달 방법 사용
- ➔ Side-effect를 허용하지 않음
 - > In mode의 parameter 전달 방법 사용
- ➔ Side-effect는 프로그램에 부정적인 경우가 많으므로, side-effect 발생을 억제하는 것이 바람직함



| Return될 수 있는 값의 Type

- ➔ 원칙적으로 first-class object만 return할 수 있다.
- ➔ 언어마다 함수가 return할 수 있는 값의 type이 제한되어 있다.

C : 배열과 함수를 제외한 모든 type이 가능

- 배열과 함수는 first-class object가 아님
- 배열과 함수는 pointer를 이용해서 return할 수 있다.

C++ : C와 비슷

- class를 return할 수 있다. (class 객체가 first-class object임)

C#, Java : 모든 type을 return할 수 있음

- Method는 return할 수 없음 (method는 first-class object가 아님)

Ada, Python, Ruby, Lua : 모든 type을 return할 수 있음

- Python, Ruby, Lua에서 함수는 first-class object임
- Ada의 경우, 함수는 type이 아님 (함수는 first-class object가 아님)

대부분의 함수형 언어(functional language) : 모든 type을 return할 수 있음

- 대부분의 type(함수포함)이 first-class object임



| Return 값의 개수

➔ 함수는 보통 1개의 값을 return한다.

```
int foo(...) { ... return 35; }  
  
x = foo(...);
```

➔ 여러 개의 값을 return하는 언어도 있다.

Ruby

```
def goo(...)  
  ... return 47, sum, average  
end  
n, total, mean = goo(...)
```

Lua

```
function hoo(...)  
  ... return 47, sum, average  
end  
n, total, mean = hoo(...)
```



| 사용자 정의 Overloaded Operator(중복 연산자)

➔ 모든 언어는 기본적으로 overloaded operator를 지원한다. (+, -, *, /, ...)

```
int * int,    float * float,    double * double
```

➔ 프로그래머가 overloaded operator를 정의할 수 있다.

- ▶ Overloaded operator는 함수를 이용해서 정의한다.
- ▶ 기본적으로 지원되는 연산자에 다른 의미를 부여한다.

```
complex * complex,    rational * rational, ...
```

- ▶ 사용자 정의 overloaded operator의 우선순위(precedence), 결합법칙(associative rule) 등은 원래의 연산자와 동일하다.
- ▶ Writability 향상, readability 저하
- ▶ 사용자 정의 overloaded operator를 허용하는 언어: C++, Ada, Python, Ruby

예

(C++): vector의 내적(inner product, dot product)

```
typedef    struct { double x, y, z; }    Vec3D;
```

```
double operator*(const Vec3D &va, const Vec3D &vb)
{    return va.x*vb.x + va.y*vb.y + va.z*vb.z;    }
```

```
dotP = vec1 * vec2;    // vec1,vec2: Vec3D, dotP: double
```



| 사용자 정의 Overloaded Operator 의 평가

- ➔ 연산자의 원래 의미와 연관이 없는 의미로 정의하면 혼란을 초래함
- ➔ 소프트웨어의 여러 module들이 동일한 연산자를 다른 의미로 정의했다면, module 통합 전에 overloaded operator의 의미를 통일시키는 것이 필요함
- ➔ Overloaded operator의 사용 여부는 readability를 고려해야 함
 - ▶ 한 operator(연산자)가 다른 type의 operand(피연산자)에 대해서 자주 사용된다면, 함수를 사용하는 것이 readability에 더 좋다.

예

```
a, b: vector type  c: scalar type  *: vector의 내적 연산  
c = a * b          // a, b를 scalar 변수로 인식하기 쉽다.  
→ c = dotProduct(a, b)와 같이 사용하는 것이 가독성에 좋음
```



| Coroutines

특수한 형태의 Subprogram

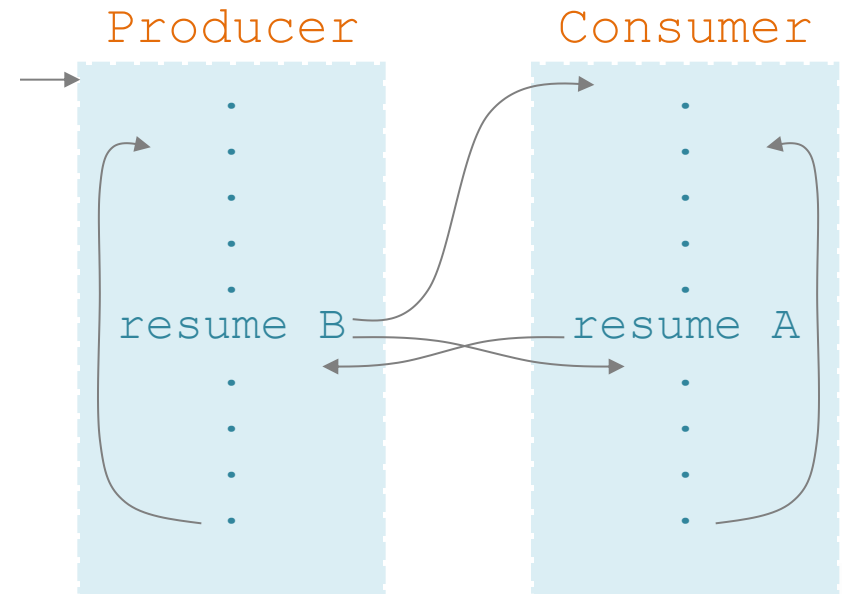
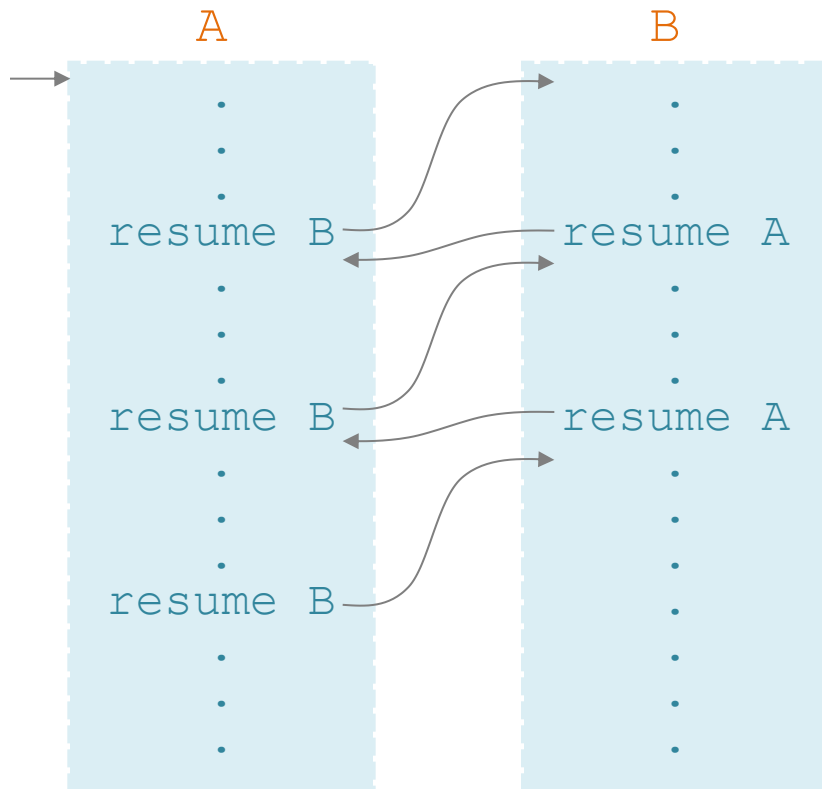
- ➔ 실행 제어 방식을 대칭적 단위 제어 모델(symmetric unit control model)이라고 한다.
 - ▶ 여러 개의 subprogram이 서로 대등한 관계임
 - > 여러 개의 subprogram 각각이 호출자(caller)가 되기도 하며, 피호출자(callee)가 되기도 함.
 - > Coroutine의 호출(call) 및 복귀(return)는 resume 명령에 의해서 이루어진다.
 - ▶ 자체에 의해서 제어되는 다중 진입 지점(entry)을 갖는다.
 - > 첫 줄 (처음 호출될 때)
 - > Resume 명령의 다음 줄
- ➔ 여러 module들이 의사-병렬실행(quasi-concurrent execution) 한다.
 - ▶ 주어진 시점에서 단 한 개의 coroutine만 실행된다.
 - ▶ 실행중인 coroutine들이 병렬로 실행되는 것처럼 보인다.

Coroutine의 적용

- ➔ 여러 module들이 서로 resume하면서 반복적으로 실행하는데 적합
 - ▶ 생산자-소비자(producer-consumer) 문제



| Coroutine의 실행 도해



평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?
총 2문제가 있습니다.

START



평가하기 1

1. 수학적 함수의 참조 투명성(referential transparency)에 대응되는 프로그래밍언어의 개념은 무엇인가?
 - a. Function이 이해하기 쉽다.
 - b. Function의 protocol만 알고 있으면 function을 이용할 수 있다.
 - c. Function이 side-effect를 유발하지 않는다.
 - d. Function의 실행환경을 프로그램 code만 봐도 알 수 있다.

확인



평가하기 2

2. Overloaded operator에 관한 설명 중 옳바르지 않은 것은?

- a. Overloaded operator는 사용자가 정의한 경우 이외에는 없다.
- b. 사용자 정의 overloaded operator의 연산 우선순위는 원래 연산자의 우선순위와 같다.
- c. 사용자 정의 overloaded operator를 허용하지 않는 언어도 있다.
- d. 사용자 정의 overloaded operator는 함수를 정의 하는 방법으로 정의한다.

확인



평가하기 3

3. 다음에 열거한 것은 subprogram들이 일반적으로 가지고 있는 특성이다. 이들 중에서 coroutine의 특성이 아닌 것은?
- a. Subprogram의 실행을 시작하는 진입 지점이 하나이다.
 - b. Subprogram의 실행이 끝나면 제어는 호출자(caller)에게 넘어간다.
 - c. 호출된 subprogram이 실행을 끝낼 때까지 호출자(caller)는 실행을 유보한다.
 - d. Subprogram이 실행을 완료하지 않은 상황에서 자신을 호출한 subprogram을 호출할 수 있다.

확인



정리하기

➡ 함수 설계 시 고려 사항

- ▶ Parameter로 인한 side-effect 발생 허용 여부
- ▶ Return될 수 값의 type
- ▶ Return할 수 있는 값(return value)의 개수
- ▶ 사용자 정의 overloaded operator 허용 여부

➡ 사용자 정의 Overloaded operator

- ▶ Overloaded operator 는 함수를 이용해서 정의한다.
- ▶ 기본적으로 지원되는 연산자에 다른 의미를 부여한다.
- ▶ 사용자 정의 Overloaded operator의 우선순위(precedence), 결합법칙(associative rule) 등은 원래의 연산자와 동일하다.

➡ Coroutine

- ▶ 대칭적 단위 제어 모델(symmetric unit control model)
- ▶ 여러 module들이 의사-병렬실행(quasi-concurrent execution)



“ 강의³를 마치겠습니다. 수고하셨습니다. ”

