Subprograms(부프로그램)

프로시저

국민대학교 컴퓨터공학부
강 승 식

Topics

- Fundamentals of Subprograms
- Design Issues for Subprograms
- Local Referencing Environments
- Parameter-Passing Methods
- Parameters That Are Subprograms
- Calling Subprograms Indirectly
- Overloaded Subprograms
- Generic Subprograms
- Design Issues for Functions
- User-Defined Overloaded Operators
- Closures
- Coroutines

Fundamentals of Subprograms

- Each subprogram has a single entry point
- The calling program is suspended during execution of the called subprogram
- Control always returns to the caller when the called subprogram's execution terminates

함수 콜 한놈은 멈춰있음
콜 당한놈이 실행됨 함수 끝나면 반드시
콜한놈한테 돌아와 주어야함

Subprogram Definition

- A *subprogram definition*
  - describes the interface to and the actions of the subprogram abstraction

- In Python
  - Function definitions are executable
  - In all other languages, they are non-executable

문장라인으로 번역해서 실행가능
다른언어들은 그렇지 않음

- In Ruby
  - Function definitions can appear either in or outside of class definitions
    - If outside, they are methods of `Object`
    - They can be called without an object, like a function

루비에서는 함수정의가
클래스 안에서 가능
밖에서도 가능

자바에서는 클래스 밖에서는 함수정의 불가능

- In Lua, all functions are anonymous

Lua는 모든 함수들에 대한 이름이 없음
(우리가 쓸일은 없음)

- *subprogram call*
  - An explicit request that the subprogram be executed
- *subprogram header*
  - Subprogram name and the formal parameters
- *parameter profile* (aka *signature*)
  - The number, order, and types of its parameters
- *protocol* 정해진규약 외부와의 약속 = 프로토콜
  - Subprogram's parameter profile and its return type
- *prototype* 함수내에서 알맹이 껍데기만 파일 체크할때 타입체크를위해 파일체크할땐 프로토타입 씀
  - Function declarations in C and C++
- *formal parameter, actual parameter*

Actual/Formal Parameter Correspondence

ABC라 하면
positional이라는 원칙에 의해
10,20,30 이 각각 A,B,C에 매칭이 되라 약속한 방법

- Positional
  - The binding of actual parameters to formal parameters is by position
  - Safe and effective
- Keyword 순서무관,A=10, B=20, C=30 순서 무관하게 써줌 ->keyword parameter
  - The name of the formal parameter is specified with the actual parameter
  - *Advantage*: Parameters can appear in any order
  - *Disadvantage*: User must know the formal parameter's names

인자가 많을때 예를들어 15개
일때 12개는 default 값넣고
ABC만 10 20 30 넣고싶을때
keyword 쓰는게 적합함

Formal Parameter Default Values

- In certain languages (e.g., C++, Python, Ruby, Ada, PHP), formal parameters can have default values
  - In C++, default parameters must appear last

C++은 맨뒷부분 값 안주면 default 값 주는데
이 default 값은 함수정의할때 명시되어있음

- Variable numbers of parameters

가변적인 갯수

Procedures and Functions

input sort output 이렇게 나뉘는게 procedure
return value를 갖는게 목적이 아님
인자값만 바꿔서 실행하면 됨 이런목적일때 씀

- *Procedures*
  - collection of statements that define parameterized computations
- *Functions*

수학에서의 function y=fx

  - structurally resemble procedures but are semantically modeled on mathematical functions

c언어에서는 별차이가 없다고 생각하여 function으로 통합함
이 두개가 merge되었음

Design Issues for Subprograms

지역변수가 정적이냐 동적이냐?

- Are local variables static or dynamic?
- Subprogram definition can be nested?

소트함수안에서
새끼함수 정의 가능하냐?
폼나는 언어(파스칼)는 함수안에 함수정의가능함
프로그램 이름있고 선언문있고 실행부분은 begin end

- Parameter passing methods

인자전달방식 (formal parameta, actual parameta)

- Are parameter types checked?

c언어의 모든 함수는 평등
함수안에 종속된 함수 허용하지 않음

- Referencing environment of a passed subprogram?
- Can subprograms be overloaded?
- Can subprogram be generic?

Local Referencing Environments

그 함수에서만 필요한거

- In most languages, locals are stack dynamic
- In C-based languages, locals can be declared `static`
- The methods of C++, Java, Python, and C# only have stack dynamic locals

메모리를 런타임 스태틱에서받아오는게아니라
처음 프로그램 시작할때 지역변수 static or global
메모리 확보를 한다. 함수가 종료될때
메모리 반납
지역변수에 메모리를 너무 많이 할당하면 안됨. 왜냐면
비효율적이다. Stack overflow 발생할수있음(팩토리얼같이
recursive 함수만들때) 이럴땐
전역변수로 선언해주면 해결 할 수 있다.

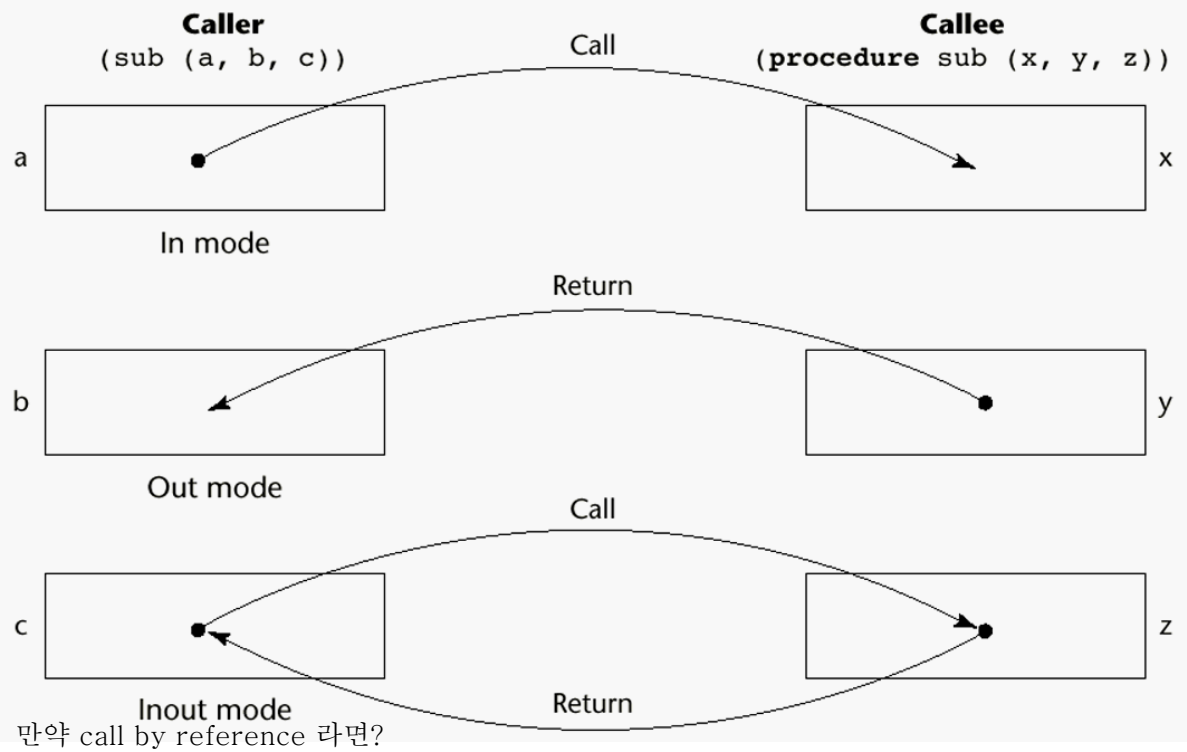
- In Lua
  - All implicitly declared variables are global
  - Local variables are declared with `local` and are stack dynamic

메모리얼로케이션 heap에서
malloc calloc
new delete

Parameter Passing

인자의 기능 a : 전달할 목적
b : 전달할목적 x 그 값중의 하나를 b로 받아오고싶을 때 그런목적
c: 전달하고 서브안에서 실행을하고 그게 끝나면(함수실행 끝나면) 받아오기도 할거임
목적이 다른 인자들

- In mode, Out mode, Inout mode



call by value는 메모리가 별도로 생김
카피본

만약 call by reference 라면?
메모리 낭비가 없이 아예 z라는 메모리를 c메모리를 가르키게함 카피를 2번할 번거로움 x
에일리어씽 기능을 이용한거임 aliasiy 임의 뭔가가 있으면 이름만 가르키고있음
즉슨 실체는 하나인데 이름이 2개이상 인걸 aliasy 라고 한다

Pass-by-Value (In Mode)

- The value of the actual parameter is used to initialize the corresponding formal parameter

- Normally implemented by copying
- Can be implemented by transmitting an access path with write protection

Out Mode, Inout mode

- Pass-by-Result (Out Mode)
  - Potential problems: `sub(p1, p1);`
- Pass-by-Value-Result (Inout Mode)
- Pass-by-Reference (Inout Mode)
  - Pass an access path
  - Passing process is efficient (no copying and no duplicated storage)
  - Potentials for unwanted side effects
  - Unwanted aliases

    `fun(total, total); fun(list[i], list[j]; fun(list[i], i);`
- Pass-by-Name (Inout Mode)
  - By textual substitution

ada가 call by value result

= call by address 주소전달방식 c++
매우 특별한 경우이긴하지만
call by value, reference 결과가 다른 경우가 있음

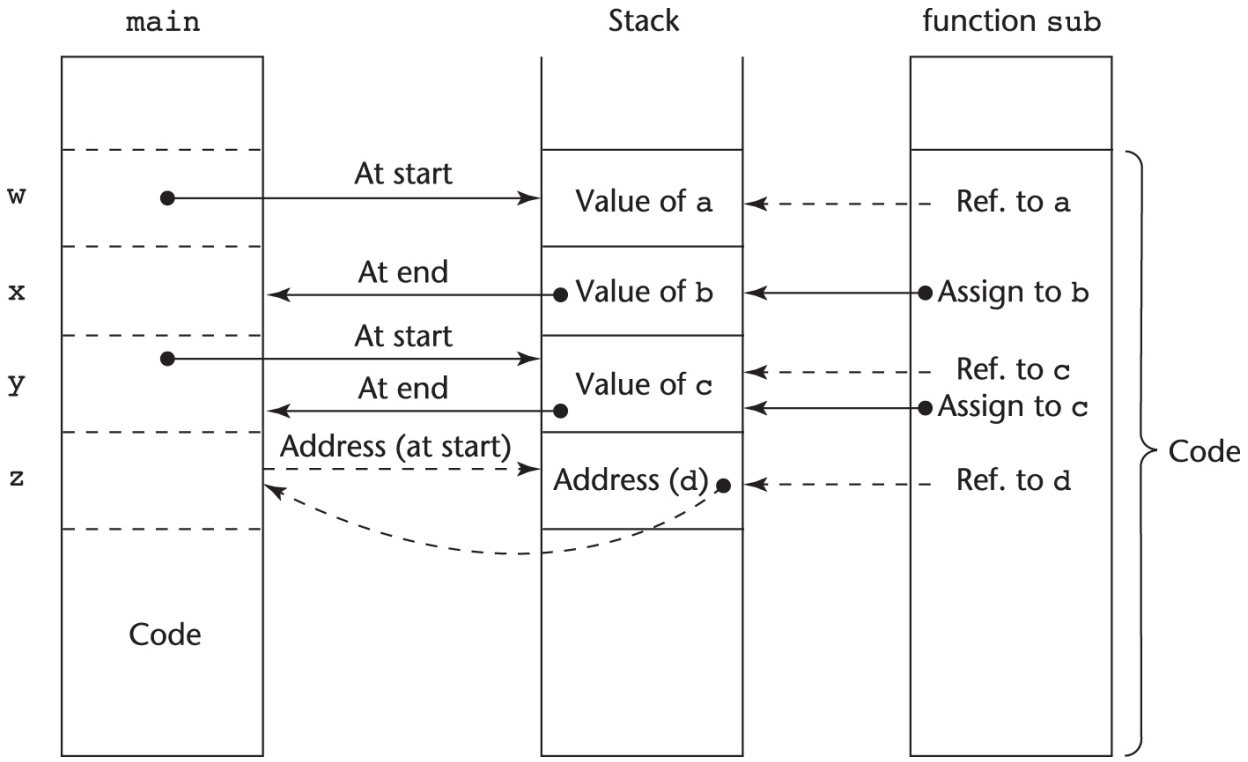
있는 그대로를 대치

Function call in main: sub(w, x, y, z)

c언어에는 call by reference가 없으므로 포인터를통해서 address를 가르키게 한다
실제로는 포인터 변수도 value 값임. 즉슨 포인터를 통해 편법으로 call by reference를 구현한다.

- pass w by value, x by result, y by value-result, z by reference

call by name 은 너무 어려워서 안함



Parameter Passing of Major Languages

- C — 받는건 편법으로 포인터를 이용해서 받는다
  - Pass-by-value
  - Pass-by-reference is achieved by using pointers as parameters
- C++ — C++은 원칙적으로는 포인터를 쓰지말고 레퍼런스를 사용해야함
  - A special pointer type called reference type for pass-by-reference
- Java — 오브젝트들은 call by reference 단순변수들은 call by value
  - All parameters are passed are passed by value
  - Object parameters are passed by reference
- Ada
  - Three semantics modes of parameter transmission: `in, out, in out`; `in` is the default mode — call by value, call by result, 둘다 합쳐놓은거
  - Formal parameters declared `out` can be assigned but not referenced; those declared `in` can be referenced but not assigned; `in out` parameters can be referenced and assigned

- Fortran 95+ in, out
  - Parameters can be declared to be in, out, or inout mode
- C# actual parameter 앞에 ref
  - Default method: pass-by-value
    - Pass-by-reference is specified by preceding both a formal parameter and its actual parameter with `ref`
- PHP: very similar to C#, except that either the actual or the formal parameter can specify ref
- Perl: all actual parameters are implicitly placed in a predefined array named @_
- Python and Ruby use pass-by-assignment (all data values are objects); the actual is assigned to the formal

call by value 란 말 쓰기 애매함

Type Checking Parameters

- Considered very important for reliability

- FORTRAN 77 and original C: none
- Pascal, FORTRAN 90+, Java, and Ada: it is always required
- ANSI C and C++: choice is made by the user
  - Prototypes

사용자가선택하게끔(프로토타입쓰는걸권장함 타입체크를 하기 위해서)

- Relatively new languages Perl, JavaScript, and PHP do not require type checking
- In Python and Ruby, variables do not have types (objects do), so parameter type checking is not possible

파이썬 루비같은경우느 타입이 없기 때문에 타입 체크 불가능

Multidimensional Arrays as Parameters

C언어 배열 타입 1차원만있음 다차원은 없음
그렇다면 int a[10][20][30]; 은?? 사실상 일차원인데
무늬만 3차원 배열임 int a[6000]; 이랑 같음
받을때(int x[]) 배열크기가 달라도 전달받을수있음
C언어는 매우 자유로운 언어임그러나 단점으론 이것으로 인해 오류 (버그) 가 많이 발생 한다,

- C, C++ 배열에 대한 포인터 전달함
  - pass a pointer to the array and the sizes of the dimensions as other parameters
- Ada – not a problem
  Ads는 엄격하게 되어있음,
  - Constrained arrays – size is part of the array's type
  - Unconstrained arrays - declared size is part of the object declaration
- Java, C# 자바나 c#은 배열이 오브젝트, 1차원밖에 허용 안함. 배열의 배열
  - Arrays are objects; all single-dimensioned, but the elements can be arrays
  - Each array inherits a named constant (`length` in Java, `Length` in C#)

Parameters that are Subprogram Names

함수이름을 함수 인자로 전달할때?

c언어의 퀵소트일때

- Issues:
  - Are parameter types checked?
  - What is the correct referencing environment for a subprogram that was sent as a parameter?
- Referencing Environment
  - *Shallow binding*: The environment of the call statement that enacts the passed subprogram
    - Most natural for dynamic-scoped languages
  - *Deep binding*: The environment of the definition of the passed subprogram
    - Most natural for static-scoped languages
  - *Ad hoc binding*: The environment of the call statement that passed the subprogram

Overloaded Subprograms

인자갯수를 보고, 타입을 봐야함 – signature 시그네이쳐라고 함

- Same name as another subprogram in the same referencing environment
- C++, Java, C#, and Ada include predefined overloaded subprograms
- In Ada, the **return type** of an overloaded function can be used to disambiguate calls

Ada에서는 return type 만 다른것도 오버로딩을 허용해줌
C++에선 안됨

Generic Subprograms

함수를 생성하는 함수
예를들어 sort 함수에서 타입에 따라 함수를 다 만들어야하느냐?
껍데기를 만들자. 하나만
이걸 지원하면 generic function
마치 C++의 template처럼

- A *generic* or *polymorphic subprogram* takes parameters of different types on different activations
- Overloaded subprograms provide *ad hoc polymorphism*

- *Subtype polymorphism* means that a variable of type T can access any object of type T or any type derived from T (OOP languages)
- A subprogram that takes a generic parameter provides *parametric polymorphism*
  - A cheap compile-time substitute for dynamic binding

- C++

```
template <class Type>
    Type max(Type first, Type second) {
    return first > second ? first : second;
    }
```

Design Issues for Functions

오리지날 펑션 side effects 해결하려면
in-mode만 지원하면됨

- Are side effects allowed?
  - Parameters should always be in-mode to reduce side effect (like Ada)
- What types of return values are allowed?
  - Most imperative languages restrict the return types
  - C allows any type except arrays and functions c언어에서 return 배열은 안됨 대신 포인터 리턴 가능 함수도 리턴안댐
  - C++ is like C but also allows user-defined types C++도 C랑 동일
  - Ada subprograms can return any type (but Ada subprograms are not types, so they cannot be returned) ada의 서브프로그램은 타입이 아니라서 return 안댐
  - Java and C# methods can return any type (but because methods are not types, they cannot be returned) 메서드는 타입이 아니라 리턴 x
  - Python and Ruby treat methods as first-class objects, so they can be returned, as well as any other class 모든 클래스를 리턴할수 있음
  - Lua allows functions to return multiple values

펑션을 리턴할수있음

Closures

값을 주어 값에 따른 함수를 만드는것을 closures 이라한다
javascript 에서 유용하게 잘 쓰임

- A *closure* is a subprogram and the referencing environment where it was defined
- A JavaScript closure:

```
function makeAdder(x) {
  return function(y) {return x + y;}
}
...
var add10 = makeAdder(10);
var add5 = makeAdder(5);
document.write("add 10 to 20: " + add10(20) + "<br />");
document.write("add 5 to 20: " + add5(20) + "<br />");
```

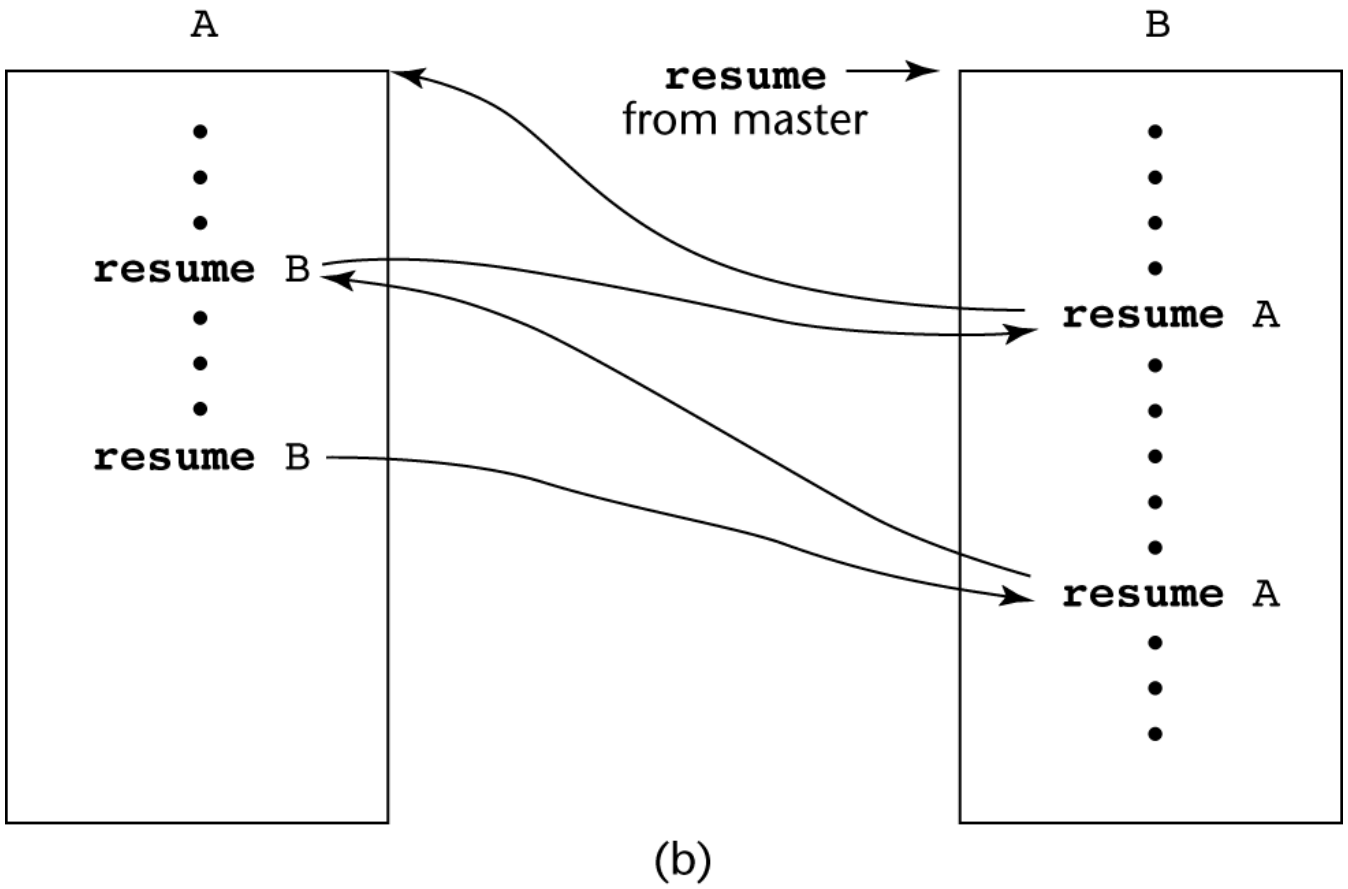
- The closure is the anonymous function returned by `makeAdder`

Coroutines

특이한함수 멀티플 엔트리 포인터
함수호출 하는게 동등함
원래는 함수 호출하는놈 당하는놈인데..\
multiple entries 설명 자세하게해줌 찾아봐 졸아서 못들음

- A *coroutine* is a subprogram that has multiple entries and controls them itself – supported directly in Lua
- Also called *symmetric control:* caller and called coroutines are on a more equal basis
- A coroutine call is named a *resume*
- The first resume of a coroutine is to its beginning, but subsequent calls enter at the point just after the last executed statement in the coroutine
- Coroutines repeatedly resume each other, possibly forever
- Coroutines provide *quasi-concurrent execution* of program units (the coroutines); their execution is interleaved, but not overlapped

Coroutines Illustrated: Possible Execution Controls



Summary

- Subprograms can be either functions or procedures
- Local variables in subprograms can be stack-dynamic or static
- Three models of parameter passing: in mode, out mode, and inout mode
- Some languages allow operator overloading
- Subprograms can be generic
- A closure is a subprogram and its ref. environment
- A coroutine is a special subprogram with multiple entries