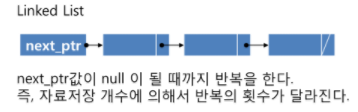
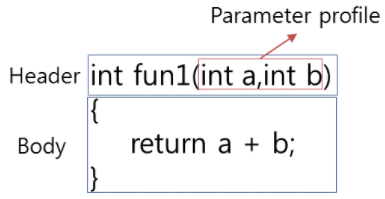
**Statement-Level Control Structures**

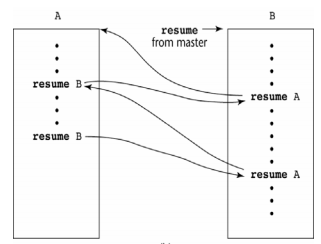
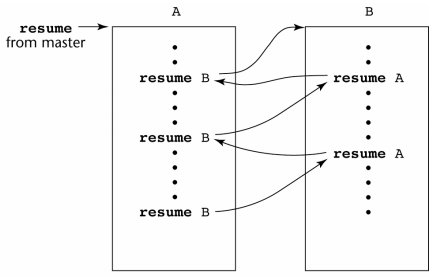
* 문장 수준에서 일어나는 제어 구문
* 함수형 언어의 expression과 function 사이의 실행 과정은 다른 expression과 function에 의해 제어된다. 이는 명령형 언어의 control statement와 유사하다.
* Control structure는 Control statement와 그것을 실행하는 statement의 모음이다.
* Multiple Selection Construct
  + Switch-case Statement
  + Else-if clause
* Iterative Statement [ Iteration or recursion ]
  + Iteration Control statement
    - 제어 방법 = Counter [ for ] or Logic [ while ]
    - 제어를 체크하는 위치가 앞일 경우 pre-test 뒤일 경우 post-test
  + User-Located Loop Control
    - 특정 경우에 Loop를 나가거나, 이하를 생략 후 다시 반복하라는 것을 정의
    - Break, Continue in Java, last in Perl
  + Iteration Based on Data Structure
    - Data Structure에서 해당위치에 값이 있는가 없는가에 의한 판단을 실시하여 순환 발생
    - In C



* + - Range, foreach in C# & Perl, block in Ruby

**Subprogram**

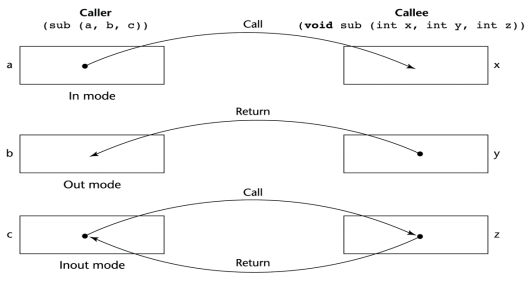
* Subprogram abstraction에 대한 interface와 action을 설명한다.
* 특성
  + **Single entry Point** : 서브프로그램을 호출 시 서브프로그램의 시작점이 항상 일정
  + Caller : 서브프로그램을 호출하는 것
  + Callee : 호출 되는 서브 프로그램
* 정의
* 
* Header + Body로 이뤄져 있다.
* Call : 서브프로그램을 호출하는 행동
* **Parameter profile** : 서브프로그램을 작동하는데 필요한 파라미터의 수, 순서, 타입 정의
* Subprogram의 종류
  + **Procedure** : 매개 변수화 된 계산을 정의하는 문장의 집합 [ ex) sort ]
  + **Function** : 프로시저와 모습은 비슷하나 수학 함수가 모델 [ ex) power(2,3) ]
  + Overloaded subprogram
  + Generic Subprogram : 다양한 타입의 매개변수를 허용하는 subprogram [ template ]
* **Prototype** : return type(protocol) + parameter profile
* 파라미터의 종류
  + **Formal parameter** : 더미로 subprogram call 시 서브프로그램의 지역변수로 생성
  + **Actual parameter** : 실제 넘기는 값으로 메인 프로그램에서 가지고 있는 변수
  + **Position paramete**r : command line에서의 parameter ( 순서가 매우 중요 )
  + **Keyword parameter** : parameter name = <expr> 형식으로 parameter 이름 지정
* **Overloaded Subprogram**
  + **Function Overloading, Polymorphic subprogram**라고 불린다.
  + **Ad hoc polymorphism**이라고 불리는 polymorphism을 지원한다.
  + C언어는 지원하지 않음
* **Generic Subprogram**
  + 다른 타입의 데이터에 동일한 동작을 하는 다른 subprogram의 생성을 줄일 수 있다.
  + **OOP**는 **Subtype polymorphism**을 지원한다**.**
  + **Parametric polymorphism**은 subprogram이 **generic(template) parameters**를 가짐으로서 제공된다.
* **Coroutine**
  + **suspend/resume** 기능이 추가된 subprogram
  + caller가 함수를 call하고, 함수가 caller에게 값을 return하면서 종료하는 것에 더해 return하는 대신 suspend하면 caller가 나중에 resume하여 중단된 지점부터 실행을 이어갈 수 있다.



**Local variable, Passing Model**

* **Local Variable**
  + 대부분의 언어는 상대위치를 저장 후 call시 메모리를 할당하는 stack-dynamic 사용
  + 장점
    - 메모리의 Recursion 가능
    - 기억 공간을 다른 서브프로그램과 공유 가능 = 메모리 절약

( -> A subprogram이 메모리를 반납 후 B subprogram에서 사용가능 )

* + 단점
    - 지속적인 할당과 회수로 인해 overhead 발생
    - 직접적이지 않은 주소 참조로 속도 저하
    - 이전에 subprogram에서 수행한 값을 같은 subprogram과 공유 불가
  + Static keyword
    - 3번 단점의 극복을 위해 제공된 방법
    - Subprogram 실행여부와 관계없이 특정 메모리를 점유하여 지속적으로 값 유지
* **Parameter-Passing Method**
  + **Semantic Models**
    - **In-mode** : 서브프로그램 호출 시 actual parameter의 값을 전달 받는다..
    - **Out-mode** : 서브프로그램을 호출하고 종료 시 formal에서 actual로 값을 넘김
    - **Inout-mode** : in-mode, out-mode 모두 가능한 mode
    - 
* **Parameter-Passing Model**
  + Pass-by-**value [in-mode]**, Call by value
    - Subprogram call 시 caller의 값을 callee로 전달
    - 주소 값 전달 시 in-mode 유지를 위해 only read로 제공해야 한다.
    - 복사하여 사용하므로, 메모리효율이 낮고, 시간비용이 높다.
  + Pass-by-**result [out-mode]**, call by result
    - callee에서 caller로만 전달이 가능
    - 단점
      * 단지 전달만 이루어 지므로 path by value와 같은 단점 발생
      * 순서에 따라 다른 값이 나올 수 있다.
  + Pass-by-**value-result [in-out-mode]**, call by copy, call by value-result
    - Subprogram 시작 시 in-mode 종료 시 out-mode 적용
  + Pass-by-reference [inout-mode], pass by sharing, call by reference, call by sharing
    - Actual parameter의 주소 값을 Formal parameter로 전달해 준다.
    - 파라미터 전달 시 메모리 카피로 인한 시간과 메모리 낭비 없음
    - 단점
      * Access가 느려지게 된다.
      * 답이 달라질 수 있다.,
  + Pass by **name [multiple mode]**
    - Multiple mode : 3가지 모드 모두 가능하다
* **Calling Subprogram Indirectly**
  + **Function pointer**
  + C# **delegate**
    - 메소드를 변수처럼 사용할 수 있게 만들어준다. 따라서 Delegate가 참조하는 메소드를 바꿀 수 있고 매개변수로 사용하여 메소드를 넘겨줄 수 있다.