프로그래밍 언어론

Parameters
(매개변수)

컴퓨터공학과
이만호

충남대학교
CHUNGNAM NATIONAL UNIVERSITY

Parameters

학습목표

Parameter(Parameter) 전달 방법의 여러 가지 의미적 모델과 그 구현 모델에 대해서 학습한다.

학습내용

- Parameter 전달 Model
- Parameter의 Type check
- 다차원 배열의 Parameter

목 차

- 알고가기
- 학습하기
  - Parameter 전달의 의미적 Model
  - Parameter 전달 방법
  - Parameter의 Type checking
  - 다차원 배열의 Parameter
- 평가하기
- 정리하기

알고가기

Subprogram에 관한 설명으로 적절하지 않은 것은?

(a) Subprogram은 동일한 유형의 문제를 여러 번 해결하는데 유용하다.

(b) Subprogram은 실행하는데 필요한 어떤 값을 parameter로 전달 받아야만 한다.

(c) Subprogram을 호출할 때 actual parameter와 호출되는 subprogram의 formal parameter사이에 data가 전달될 수 있다.

(d) Subprogram을 실행한 결과는 parameter를 통해서 얻을 수도 있다.

확인

Parameter 전달의 의미적 Model – 1

Formal parameter는 다음 3개의 의미적 모델(semantic model) 중 하나이다.

In mode(입력 모드)

- Formal parameter는 actual parameter로부터 data를 받는다.

Out mode(출력 모드)

- Formal parameter는 actual parameter에 data를 전달한다.

Inout mode(입출력 모드)

1. Formal parameter는 actual parameter로부터 data를 전달받는다.
2. Formal parameter는 actual parameter에 data를 전달한다.

전달되는 data

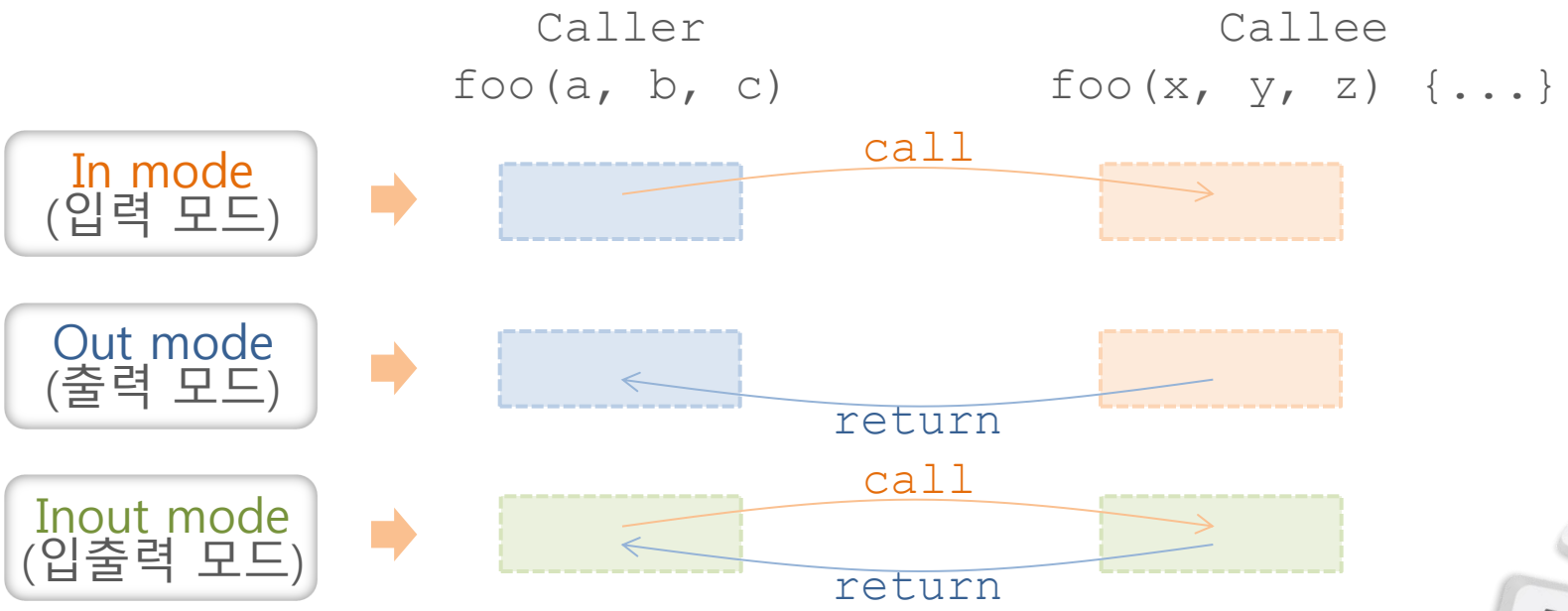
Value(값)

Access path(접근 경로)

: pointer 또는 참조(reference)

Parameter 전달의 의미적 Model – 2

```
foo(x, y, z, ...) { ... } : Callee
    in
       out
          inout
foo(a, b, c, ...)          : Caller
```



Parameter 전달 Model의 구현

In mode(입력 모드)

- Call-by-value(값-전달, pass-by-value)

Out mode(출력 모드)

- Call-by-result(결과-전달, pass-by-result)

Inout mode(입출력 모드)

- Call-by-value-result(값-결과-전달, pass-by-value-result)
- Call-by-reference(참조-전달, pass-by-reference)

Multiple mode(다중 모드)

- Call-by-name(이름-전달, pass-by-name)

Call-by-Value(값-전달)

Parameter 전달 Model : In mode(입력 모드)

Formal parameter로 전달되는 Data

- Actual parameter의 값(value) (경우에 따라서는 값이 접근 경로(access path)일 수도 있다.)

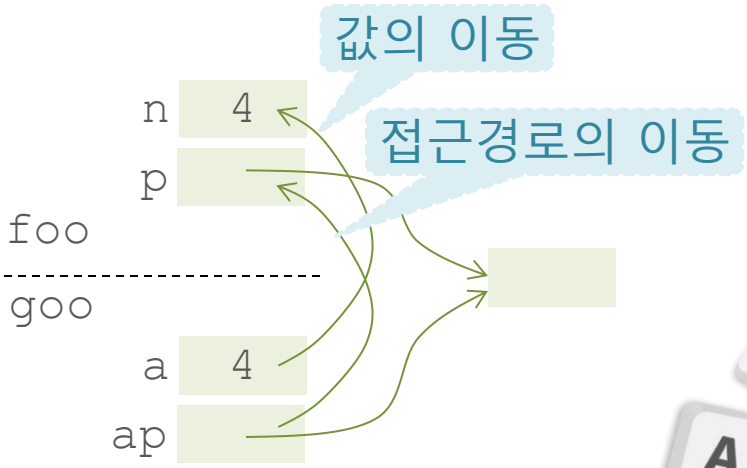
구현 방법

1. Formal parameter가 사용할 기억장소를 할당한다.
2. Formal parameter를 전달되는 actual parameter의 data로 초기화한다.
3. Formal parameter는 subprogram 내에서 지역변수처럼 사용된다.

Actual parameter로 전달되는 data : 없음

```
int foo(int n, int *p)
{ ... }

int goo(...)
{ int  a=4;
  int  *ap=(int*)malloc(...);
  ...
  ... foo(a, ap) ...
  ...
}
```



Call-by-Value의 평가

장점

- **Parameter가 scalar 변수인 경우,**
  - Formal parameter의 접근 시간이 효율적임
- **Subprogram의 실행 과정 및 결과가 호출 프로그램(caller)에 영향을 끼치지 않음**

단점

- **Parameter가 배열(array)인 경우,**
  - Actual parameter인 배열의 크기만큼 기억 공간이 추가로 필요함
  - Actual parameter인 배열의 모든 값을 formal parameter로 복사해야 함 (호출할 때)
- **접근 경로(access path)가 전달되는 경우,**
  - Subprogram에서 쓰기-방지(write-protect)를 보장해야 함 : 구현하기 어려움
  - Parameter 접근은 간접 주소(indirect addressing)를 이용함 => 접근 시간이 김

Call-by-Result(결과-전달)

Parameter 전달 Model : **Out mode**

Formal parameter로 전달되는 Data : 없음

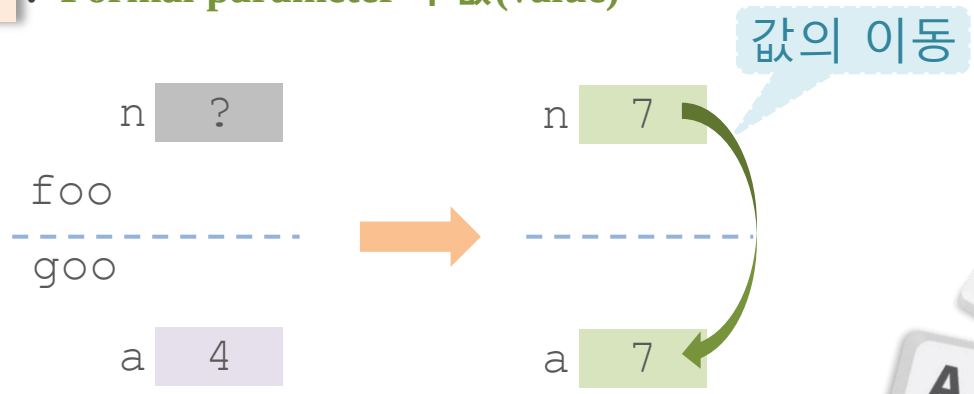
구현 방법

1. Formal parameter가 사용할 기억장소를 할당한다.
2. Formal parameter는 subprogram 내에서 지역변수처럼 사용된다.
3. Subprogram의 실행이 끝나고 호출자(caller)로 return할 때, formal parameter의 값이 actual parameter로 전달된다.

Actual parameter로 전달되는 data : **Formal parameter의 값(value)**

```
int foo(int n)
{ ... n=7; }

int goo(...)
{ int  a=4;
  ...
  ... foo(a) ...
  ...
}
```



Call-by-Result의 평가

장점 및 단점

- Call-by-value와 비슷
- Parameter 사이에 값이 전달되는 시점에 따라 약간의 차이가 있음

  값이 전달되는 시점: **호출할 때, 또는 return할 때**

장점

- Parameter가 scalar 변수인 경우,
  - Formal parameter의 접근 시간이 효율적임
- Subprogram의 실행 과정이 호출 프로그램(caller)에 영향을 끼치지 않음

단점

- Parameter가 배열(array)인 경우,
  - Actual parameter인 배열의 크기만큼 기억 공간이 추가로 필요함
  - Actual parameter로 전달되는 배열의 모든 값을 복사해야 함 (return할 때)
- 호출 프로그램(caller)에서 actual parameter로 사용된 변수의 값이 변화됨

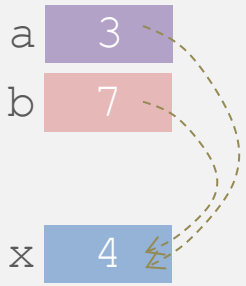
Call-by-Result의 문제점

문제점1

변수 하나가 actual parameter로 두 번 이상 사용될 때, formal parameter의 값이 actual parameter로 전달되는 순서에 따라서 결과가 달라짐

```
int foo(int a, int b)
{ a=3; b=7; }

int goo(...)
{ int  x=4;
  ...
  foo(x, x)
  ...
}
```



문제점2

Actual parameter가 배열 요소인 경우, 주소 계산 시점에 따라서 결과가 달라짐.

주소 계산 시점: 호출할 때, 또는 return할 때

```
int index=2;  //전역변수

int foo(int s)
{ index=5; s=39; }

int goo(...)
{ int  xa[10];

  foo(xa[index])
  ...  // xa[2] 또는 xa[5]
}
```

Parameter 전달 순서와 주소 평가 시점은 구현에 종속적임

→ 이식성에 문제가 있음

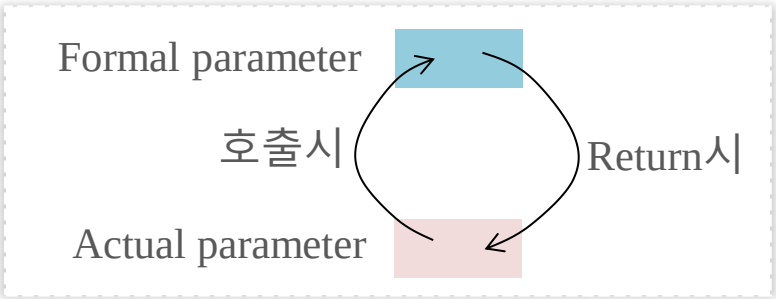
Call-by-Value-Result(값-결과-전달)

Parameter 전달 Model : Inout mode

- Call-by-value와 call-by-result가 혼합된 형태

구현 방법

1. Formal parameter가 사용할 기억장소를 할당한다. (call-by-value, call-by-result)
2. Formal parameter를 전달되는 actual parameter의 data로 초기화한다. (call-by-value)
3. Formal parameter는 subprogram 내에서 지역변수처럼 사용된다.
4. Subprogram의 실행이 끝나고 호출자(caller)로 return할 때, formal parameter의 값이 actual parameter로 전달된다. (call-by-result)



Call-by-Value-Result의 평가

- 호출할 때: call-by-value와 동일
- Return할 때: call-by-result와 동일

- Actual parameter가 상수일 경우, 상수 값이 달라질 수 있다.

```
int foo(int a) { a=8; }

int goo(...)
{ int  k;
  foo(5);  k = 5; }
```

```
int foo(int a) { a=8; }

int goo(...)
{ int  k;
  foo(5);  k = 5; }
```

Call-by-Reference(참조-전달)

Parameter 전달 Model : Inout mode

- Call-by-value와 call-by-result가 혼합된 형태

Formal parameter로 전달되는 Data

- Actual parameter의 access path(접근 경로, 주소)

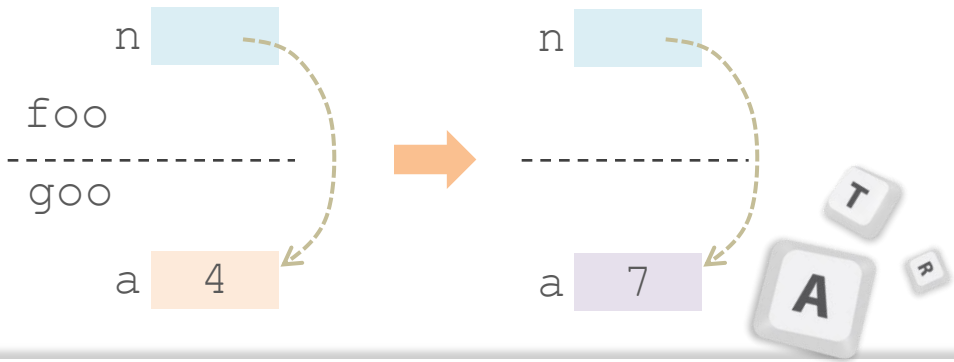
구현 방법

1. Formal parameter가 사용할 기억장소로 주소를 저장할 수 있는 공간을 할당한다.
2. Formal parameter를 전달되는 actual parameter의 주소(address)로 초기화한다. (구현하기가 가장 쉽다)
3. Subprogram 실행 과정에서, formal parameter에 저장된 주소를 통해서 호출자(caller)에서 사용되는 변수에 접근할 수 있다. (indirect addressing)

Alias 현상 발생

```
int foo(int n)
{ ... n=7; }

int goo(...)
{ int  a=4;
  ... foo(a) ...
  ...
}
```



Call-by-Reference의 평가

장점

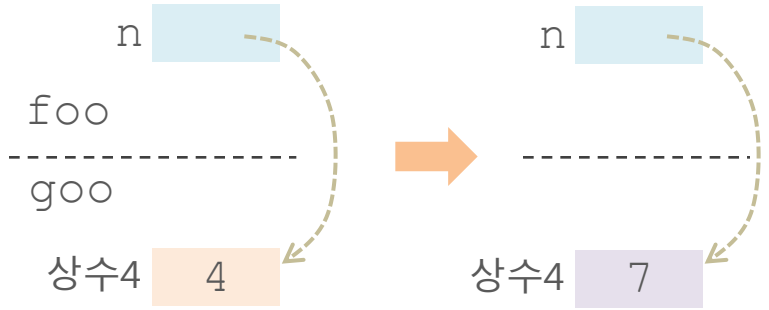
- **Parameter 전달 과정이 효율적이다.**
  - Formal parameter를 위한 기억장소는 주소를 저장할 수 있으면 됨
  - 주소 하나만 복사됨 (parameter가 배열인 경우라 할지라도)

단점

- **Formal parameter에 접근할 때,**
  - 간접 주소(indirect addressing) 사용 → 접근 시간(access time)이 오래 걸림
  - Subprogram의 비 지역변수(호출자의 지역변수)에 접근함 → side-effect 발생 가능
- Actual parameter가 상수일 경우, 상수 값이 달라질 수 있다.

```
int foo(int n)
{ ... n=7; }

int goo(...)
{ int  k;
  ... foo(4) ...
  k = 4
}
```



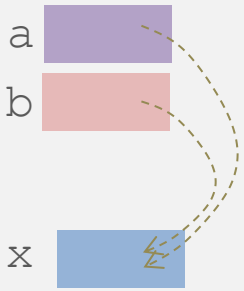
Call-by-Reference의 문제점

문제점1

변수 하나가 actual parameter로 두 번 이상 사용될 때, alias 현상 발생

```
int foo(int a, int b)
{ ... }

int goo(...)
{ int  x;
  ...
  foo(x, x)
  ...
}
```



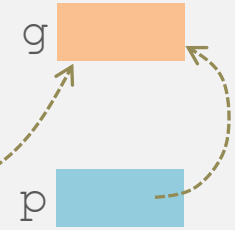
문제점2

Actual parameter전역변수로 사용될 때, alias 현상 발생

```
int g;  //전역변수

int foo(int p)
{
  ...
  /* g 접근 가능 */
}

int goo(...)
{
  foo(g)
  ...
}
```



Call-by-Name(이름-전달)

Parameter 전달 모델 : Inout mode

- Actual parameter 유형에 따라 parameter 전달 모델이 다름

Parameter 전달 방법

- Actual parameter의 이름 자체(text)가 formal parameter로 전달됨
- Formal parameter는 actual parameter의 이름 그대로 대치됨
  - Formal parameter를 대치한 actual parameter 이름은 그 상황에서 scope rule이 적용됨
  - Actual parameter의 값은 사용될 때 계산됨 (lazy evaluation/binding) --- thunk를 이용

문제점

- 매우 비효율적
- 프로그램 이해가 어려움 (나쁜 readability)

Actual parameter 유형	Parameter 전달 Model
상수	Call-by-value
Scalar 변수	Call-by-Reference
배열 요소	-
변수가 포함된 수식	-

```
int  k=3, a[9];  // 전역변수

int foo(n, x, y, z)
{ x = n;      → k = 4;
  y = z;      → a[k] = k+2;
}
int goo(...)
{ ...
  foo(4, k, a[k], k+2);
}
```

주요 언어의 Parameter 전달 방법 - 1

C

- Call-by-value
- Parameter가 pointer인 경우 call-by-reference처럼 사용될 수 있음

C++

- Call-by-value
- Call-by-reference: Formal parameter앞에 '&'를 붙임 (reference type)

```
void foo(int p1, int &p2, const int &p3) { ... }
```

C#

- Call-by-value
- Call-by-reference: Formal parameter선언과 actual parameter 앞에 "ref"를 붙임 (reference type)

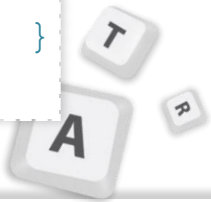
```
void foo(int p1, ref int p2) { ... }
... ...  foo(a1, ref a2); ...
```

- Out mode를 지정할 수 있다.

```
void foo(out int p) { ... }
...  foo(a); ...
```

또는

```
void foo(int p) { ... }
...  foo(out a); ...
```



주요 언어의 Parameter 전달 방법 - 2

Java

- Call-by-value
- Call-by-reference: 모든 object

Ada

- Call-by-value-result: scalar 변수
- Call-by-reference: 구조체(structured) 변수
- 각 formal parameter마다 전달 model을 지정한다.

```
procedure foo(A:in Integer; B:out Integer; C:in out Integer)
```

>> 각 formal parameter는 지정된 전달 model의 용도에 따라 사용되어야 한다.

Fortran 95

- Call-by-reference
- 각 formal parameter마다 전달 model을 지정할 수 있다.

```
Subroutine foo(A, B, C)
   Integer, Intent(In)    :: A
   Integer, Intent(Out)   :: B
   Integer, Intent(Inout) :: C
```

주요 언어의 Parameter 전달 방법 - 3

PHP : C#과 비슷

- **Call-by-value**
- **Call-by-reference: Formal parameter와 actual parameter 중 하나 또는 모두의 앞에 '&'를 붙임 (reference type)**

```
function foo(&$p) { ... }
...  foo($a); ...
```

또는

```
function foo($p) { ... }
...  foo(&$a); ...
```

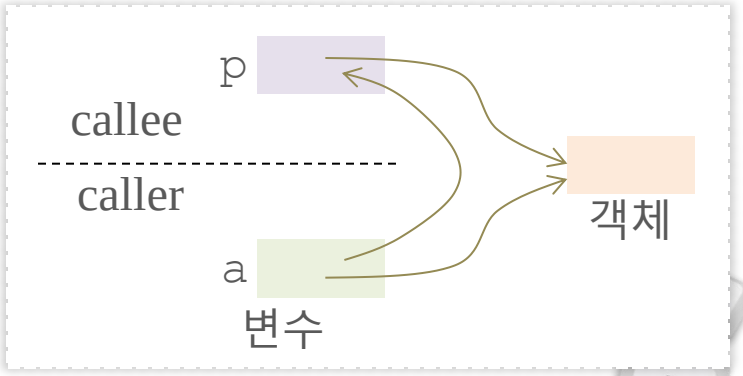
Perl

- **모든 actual parameter는 미리 정의된(pre-defined) 배열인 @_로 전달된다.**
- **Call-by-reference**

  >> Formal parameter@_의 값이 바뀌면, actual parameter의 값도 바뀐다.

Python, Ruby

- **언어의 특징**
  - 모든 data는 값 변경이 불가능한 객체(object)임
  - 모든 변수는 객체에 대한 참조(reference to object)임
- **Call-by-assignment(할당-전달)**
  - Scalar 변수: call-by-value
  - 배열: 배열 요소의 값이 변경되는 경우에는 call-by-reference



Parameter의 Type checking

Actual parameter와 Formal parameter의 type 호환성을 검사함

- 프로그램의 reliability에 매우 중요함

언어별 Parameter Type checking

- Fortran 77, original C, Perl, JavaScript, PHP : Type check를 하지 않음
- Pascal, Fortran 90, Java, Ada: Type check를 항상 수행함
- ANSI C, C++: prototype에서 type check 여부를 지정할 수 있음

```
double sin(x)
double x; // not type check
{ ... }

value = sin(n); // n: int
```

또는

```
double sin(double x)
    // do type check, and coercion
{ ... }

value = sin(n);  // n: int
```

- Python, Ruby: 변수는 type이 없으므로 type check를 할 수 없음

Parameter로서의 다차원 배열

Parameter로 전달된 다차원 배열

- Actual parameter의 크기가 일정하지 않을 수 있다.
- 컴파일러는 배열 원소의 주소 계산을 위해 주소 계산 함수가 필요하다.

다차원 배열의 주소 계산 함수

- Actual parameter의 차원과 각 차원의 크기에 따라 다르다.
- Subprogram은 actual parameter 배열의 크기 정보를 전달 받아야 한다.
- 배열의 기억정소 저장 방식에 따라 다르다.
  - 행-우선(row-major): 일반적
  - 열-우선(column-major)

다차원 배열 Parameter (C, C++)

Actual parameter 배열의 모든 차원의 크기 정보를 명시해야 함.

- Actual parameter 배열의 첫째 차원의 크기는 필요 없음
- 유연한(flexible) subprogram 작성하기가 불가능하다.

```
int foo(int a[][4][5])
{ ... a[i][j][k] ... }
```

>> Actual parameter의 크기가 "[*][4][5]"인 경우에만 사용할 수 있다.

유연한 subprogram 작성하기

- Formal parameter
  - 다차원 배열을 마치 1차원 배열인 것처럼 전달 받음
  - 다차원 배열의 첫 요소의 주소
  - 각 차원의 크기 정보 (첫째 차원의 크기 정보는 필요 없음)
- Subprogram body에서 배열 요소의 주소 계산 함수 이용

```
int foo(int *a, int n0, int n1, int n2) // a: 3-dim
{ ...      a[i*n1*n2 + j*n2 + k] ...    // n0는 필요 없음
  ... *(a + i*n1*n2 + j*n2 + k) ...
}
```

다차원 배열 Parameter (Ada)

Constrained array(크기 지정 배열)

- 배열의 크기 정보가 배열 type의 일부임
- 유연한(flexible) subprogram 작성하기가 불가능하다.

```
type Mat_CType is array (1..9, 0..6) of Integer;
function foo(a: Mat_CType) return Integer is ...
```

Unconstrained array(크기 비지정 배열)

- 배열 크기 정보가 formal parameter에 포함되지 않음
- Actual parameter의 배열 크기 정보가 호출할 때 binding됨

```
type Mat_UType is array (Integer range <>,
                         Integer range <>) of Integer;
Mat : Mat_UType(1..9, 0..6)
   // Mat'range(1): 1..9, Mat'range(2): 0..6
function goo(b: Mat_UType) return Integer is ...
   ... b'range(1) ...     // b'range(1): 1..9
   ... b'range(2) ...     // b'range(2): 0..6
   ... b[r,c] ...         // 1<=r<=9,  0<=c<=6
```

다차원 배열 Parameter (Java, C#)

배열의 특성

- 모든 배열은 객체(object)이다.
- 모든 배열은 1차원 배열이다.
- 1차원 배열의 요소는 1차원 배열일 수 있다.
- 각 배열의 크기는 이름 상수(named constant)에 저장된다.
  - length: Java
  - Length: C#

Formal parameter는 '[]'의 개수로 배열의 차원을 나타낸다.

```
int foo(int f[][]) {
    ... f.length ...        // f.length: 첫째 차원의 크기
    ... f[r].length ...     // 0 <= r <= f.length
    ... f[r][c] ...         // 0 <= c <= f[r].length
}
int goo(...) {
  int  ap[9][6], bp[7][8];
  ... foo(ap) ... foo(bp) ...
}
```

다차원 배열 Parameter (Fortran)

배열의 특성

- 배열의 기억장소 할당은 열-우선(column-major)이다.

Formal parameter로서의 배열

- 배열의 크기 정보를 parameter로 전달 받아야 한다.
- Subprogram header 다음 줄에 parameter로 전달 받은 배열의 크기 정보를 이용하여 선언되어야 한다.

```
Subroutine foo(Mat, R, C, Result)
  Integer, Intent(In) :: R, C
  Integer, Dimension(R,C), Intent(In) :: Mat
  Integer, Intent(Out) :: Result
  ...
End Subroutine foo

Subroutine goo(...)
  Integer Dimension(9,6) :: Ma
  Integer Dimension(7,8) :: Mb
  ... Call foo(Ma, 9, 6, MaR) ...
  ... Call foo(Mb, 7, 8, MbR) ...
End Subroutine goo
```

Parameter 전달 고려사항

중요한 고려 사항

- **효율성(efficiency)**
- **Parameter 전달 방향**
  - 단방향(one-way) : in mode, out mode
    - call-by-value, call-by-result
  - 양방향(two-way) : inout mode
    - call-by-reference, call-by-value-result

효율성과 Parameter 전달 방향은 서로 상충 관계이다.

- **단방향은 신뢰도가 높으나, 효율적이지 않다.**

  >> 크기가 큰 배열의 경우 call-by-value가 효율적이지 않다.

- **양방향은 신뢰도가 나쁘나, 효율적이다.**

  >> 크기가 큰 배열의 경우 call-by-reference가 효율적이다.

Parameter로서의 Subprogram

Subprogram을 Parameter로 전달하는 것이 편리할 수도 있다.

예 **삼각함수를 적분하기**

```
double Integr(double (*fun)(double), int low, int up)
{ ... (*fun)(x) ... }

void goo(...)
{ ... sina = Integr(sin, 0, 1); ...
  ... cosa = Integr(cos, 0, 1); ...
  ... tana = Integr(tan, 0, 1); ... }
```

설계 고려 사항

- **Parameter로 전달된 subprogram에 전달되는 parameter의 type check 여부**
  - Java, Ada: subprogram을 parameter로 전달하지 못함
  - C, C++: 함수 pointer를 parameter로 전달하고, 전달된 함수의 parameter type check 실행
- **Parameter로 전달된 subprogram의 실행환경(referencing environment)은 무엇?**
  - Shallow binding: 전달된 subprogram을 **호출**한 subprogram의 실행환경
    > Dynamic-scope 언어에 적합
  - Deep binding: 전달된 subprogram을 **선언**한 subprogram의 실행환경
    > Static-scope 언어에 적합

Parameter Subprogram의 실행환경

"call sub2"가 실행될 때의 실행환경

호출 순서: sub1, sub3, sub4, sub2

Shallow binding

- 전달된 subprogram을 호출한 subprogram의 실행환경
  - sub2, sub4, sub3, sub1
- 출력: 4

Deep binding

- 전달된 subprogram을 선언한 subprogram의 실행환경
  - sub2, sub1
- 출력: 1

Ad hoc binding

- 전달된 subprogram을 전달한 subprogram의 실행환경
  - sub2, sub3, sub1
- 출력: 3

```
procedure SUB1;
  var  x: integer;
  procedure SUB2;
    begin
      write('x =', x);
    end;
  procedure SUB3;
    var  x: integer;
    begin
      x := 3;
      call sub4(SUB2);
    end;
  procedure SUB4(subx);
    var  x: integer;
    begin
      x := 4;
      call subx;
    end;
  begin
    x := 1;
    call SUB3;
  end;
```

deep

ad hoc

shallow

평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?

총 3문제가 있습니다.

START

평가하기 1

1. Parameter 전달에 관한 설명이다. 적절하지 않은 것은?

a. Call-by-value에서 actual parameter가 사용하는 기억장소의 주소가 formal parameter에 복사된다.

b. Parameter가 배열일 경우, parameter 전달 시간과 기억장소 관점에서 call-by-reference가 가장 효율적이다.

c. Call-by-value-result에서, actual parameter가 배열 요소일 경우, 주소가 계산되는 시점에 따라 결과가 달라질 수 있다.

d. Call-by-name으로 전달된 actual parameter는 그 값이 필요할 때 계산이 이루어진다.

확인

평가하기 2

2. Parameter 전달 model로서 inout mode에 해당되는 parameter 전달 방법은?

a. Call-by-value

b. Call-by-result

c. Call-by-reference

d. Call-by-name

확인

평가하기 3

3. C 언어에서, parameter로 2차원 배열을 전달받는 subprogram에서 사용 방법으로 적절하지 않은 것은?

a. 첫째 차원(첨자)의 크기를 지정한다.

b. 둘째 차원(첨자)의 크기를 지정하지 않아도 된다.

c. 1차원 배열처럼 전달 받을 수 있다.

d. Pointer로 전달 받을 수 있다.

확인

정리하기

Parameter 전달의 의미적 모델(semantic model)

- In mode, Out mode, Inout mode

Parameter 전달 방법

- In mode(입력 모드)
  - Call-by-value(값-전달, pass-by-value)
- Out mode(출력 모드)
  - Call-by-result(결과-전달, pass-by-result)
- Inout mode(입출력 모드)
  - Call-by-value-result(값-결과-전달, pass-by-value-result)
  - Call-by-reference(참조-전달, pass-by-reference)
- Multiple mode(다중 모드)
  - Call-by-name(이름-전달, pass-by-name)

각 Parameter 전달 방법의 장점, 단점, 문제점

Parameter의 Type checking

- Actual parameter와 formal parameter의 type 호환성을 검사한다.

다차원 배열 Parameter

- 전달되는 다차원 배열의 배열 요소 주소 계산을 위해 모든 차원의 크기 정보가 필요하다.
- 행-우선(row-major)일 경우 첫째 차원의 크기는 필요하지 않다.

"강의를 마치겠습니다. 수고하셨습니다."