

9장 부프로그램 (1)

2020. 6. 15

순천향대학교 컴퓨터공학과

목차

- 부프로그램 개념
- 매개변수
- 부프로그램 설계시 고려사항
- 매개변수 전달 방법
- 부프로그램 매개변수
- 중복 부프로그램
- 포괄형 부프로그램

부프로그램 개념

- 추상화 도구

프로세스 추상화

문장들의 집합을
부프로그램으로
추상화시키는 수단

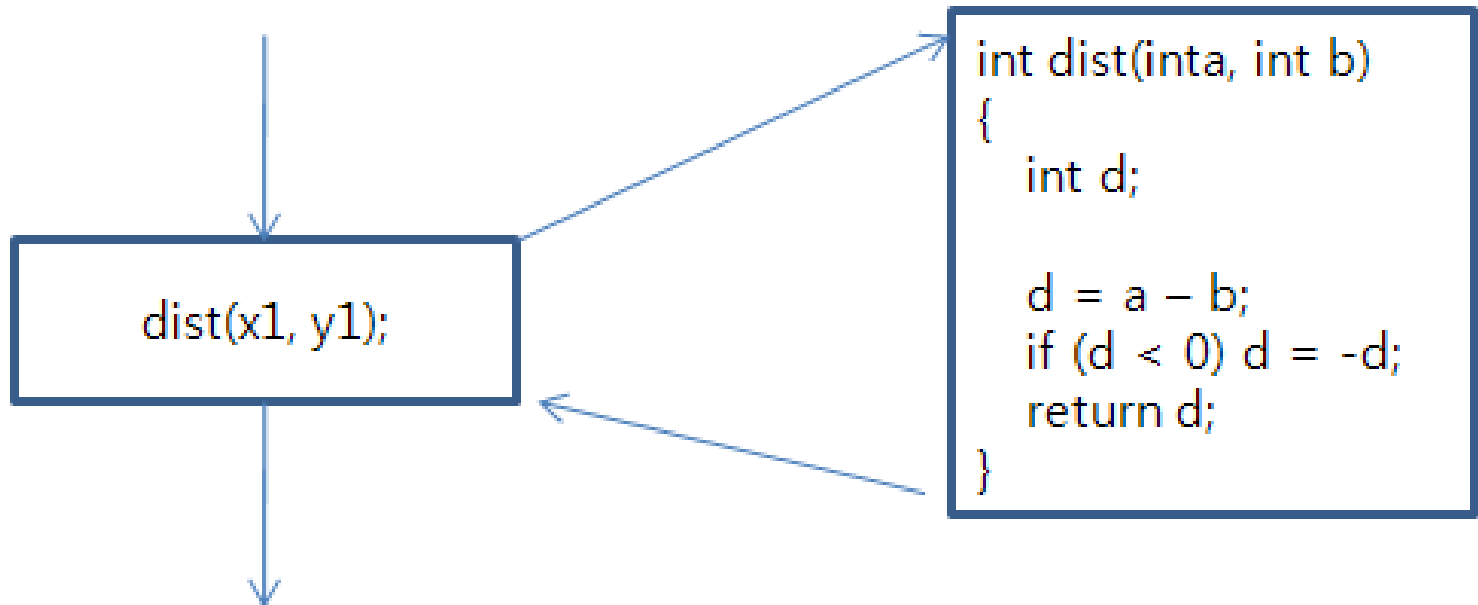
두 가지의
기본적인
추상화 도구

데이터 추상화

사용자가 새로운 데이터
타입을 정의하는 것을
허용하는 수단

부프로그램 특성

- 각 부프로그램은 단일의 진입점을 갖는다.
- 호출 프로그램 단위는 피호출 부프로그램의 실행 중에 중단된다.
- 부프로그램의 실행이 끝났을 때 제어는 항상 호출 프로그램에게 돌아간다.



부프로그램 기본 정의

- 부프로그램 정의(subprogram definition)
 - 부프로그램의 인터페이스와 동작 서술
- 부프로그램 호출(subprogram call)
 - 부프로그램이 실행되게 하는 명시적 요청
- 부프로그램 헤더(subprogram header)
 - 부프로그램 정의의 첫부분
 - 부프로그램 이름, 유형, 형식 매개변수를 포함

부프로그램 기본 정의 (2)

- 매개변수 프로파일(parameter profile)
 - 매개변수의 이름, 순서, 타입을 포함
 - 부프로그램 서명(signature)이라고 함
- 부프로그램 프로토콜(subprogram protocol)
 - 매개변수 프로파일과 반환 값 타입(존재할 경우)을 포함
- 부프로그램 선언(subprogram declaration)
 - 부프로그램의 프로토콜만 제공(몸체는 제공하지 않음)
 - 정적 타입 검사에 필요
 - C에서는 프로토타입(prototype)이라 하며, 보통 헤더 파일에 포함

부프로그램 기본 정의 (3)

- 형식 매개변수(formal parameter)
 - 부프로그램 헤더 상에 명세되고 부프로그램에서 사용되는 변수
- 실 매개변수(actual parameter)
 - 부프로그램 호출문에 나열된 값이나 주소

예제

- 다음에서 부프로그램 기본 정의를 식별하라.

```
int main()
{
    int dis(int, int);

    ...
    dist1 = dist(x1, y1);
    ....
}
```

```
int dist(int a, int b)
{
    int d;

    d = a - b;
    if (d < 0) d = -d;
    return d;
}
```

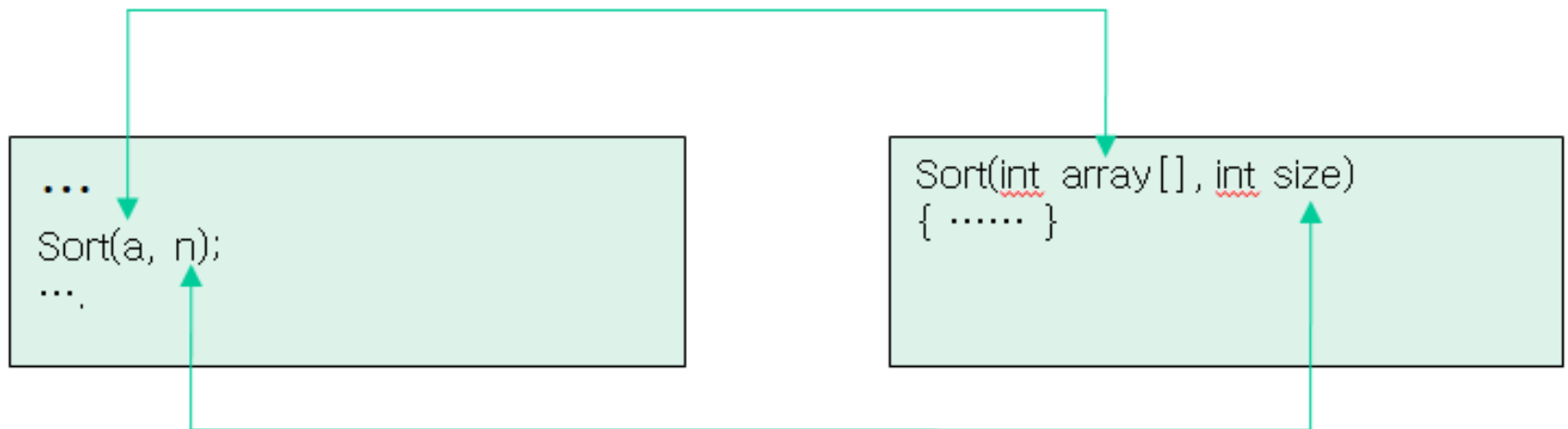
부프로그램정의
부프로그램 호출
부프로그램머리부
매개변수프로파일
프로토콜
형식매개변수
실매개변수
부프로그램선언

매개변수

- 부프로그램이 처리하는 데이터
 - 매개변수
 - 지역 변수
 - 비 지역변수 => 부작용 초래, 신뢰성 감소
- 부프로그램 호출시, 형식 매개변수는 실 매개변수에 바인딩되는데 이를 매개변수 대응(parameter correspondence)이라 한다.
- 매개변수 대응의 2가지 방법
 - 위치 기반
 - 키워드 기반

위치 매개변수

- 위치 매개변수(positional parameter)는 매개변수 대응이 단순한 위치에 기반하여 이루어짐
 - 매개변수가 나열된 위치 순서에 기반하여 대응
 - 첫번째 형식 매개변수는 첫번째 실 매개변수에 대응되고,
 - 두번째 형식 매개변수는 두번째 실 매개변수에 대응된다.
 - 안전하고 효과적인 방법



키워드 매개변수

- 키워드 매개변수(keyword parameter)는 실 매개변수에 바인딩되는 형식 매개변수의 이름이 실 매개변수와 함께 지정
 - 실 매개변수의 순서는 임의적이거나 형식 매개변수 이름 속지 필요
- In Ada,

```
Sumer(My_Length: Integer, My_Array: IntArray, My_Sum: Integer) {...}  
...
```

```
Sumer(Length => My_Length,  
      List => My_Array,  
      Sum => My_Sum);
```

키워드 매개변수(2)

- In Python,
 - 위치, 키워드 매개변수 혼합 가능
 - 키워드 매개변수가 나타난 이후에는 모두 키워드 매개변수여야 함

Sumer(length, sum = My_Sum, list = My_Array)

디폴트 매개변수

- 디폴트 값(default value)을 갖는 형식 매개변수
 - 예제 언어: C++, Python, PHP, Ada
- In Python: 생략된 매개변수 다음에는 키워드 매개변수가 온다

디폴트 매개변수



```
def compute_pay(income, exemptions = 1, tax_rate)  
...
```

```
pay = compute_pay(2000.0, tax_rate = 0.15)
```

키워드 매개변수



디폴트 매개변수 (2)

- C++에서 디폴트 매개변수는 매개변수 리스트의 마지막에 위치
 - 디폴트 매개변수가 나오면 이후 나머지 매개변수는 모두 디폴트 매개변수임
 - 키워드 매개변수를 지원하지 않음

```
float compute_pay(float income, float tax_rate,  
                  int exemptions = 1)
```

```
...
```



디폴트 매개변수

```
pay = compute_pay(2000.0, 0.15)
```

가변개수 매개변수

- 가변 개수의 매개변수를 허용한다.
- In C#
 - params 수정자로 선언된 매개변수는 가변개수 매개변수임

```
public void DisplayList(params int[] list) {  
    foreach (int next in list) {  
        console.WriteLine("Next value{0}", next);  
    }  
}  
...
```

```
int[] myList = new int[6] {2, 4, 6, 8, 10, 12};  
DisplayList(myList);  
DisplayList(2, 4, 3*x-1, 17);
```


가변개수 매개변수

가변개수 매개변수(2)

- In Python



- 매개변수 앞에 '*'가 오면 그 매개변수는 가변 개수 매개변수임
- 대응된 실매개변수는 값들의 리스트임

가변개수 매개변수



```
def myFun(arg1, *argv);  
    print("First argument:", arg1);  
    for arg in argv:  
        print("Next arguments:", arg)  
  
...  
myFun('Hello', 'Welcome', 'to', 'SCH')
```


부프로그램 유형

프로시저	함수
매개변수화된 계산을 정의하는 문장들의 집합을 추상화	수학적 함수를 모델링
새로운 문장 정의	사용자-정의 연산자 정의
가시적 <u>비지역</u> 변수나 형식 매개변수를 변경	값을 생성하여 반환
<pre>void sort(int list[], int listlen); sort(scores, 100);</pre> 	<pre>float power(float base, float exp); result = 3.4 * power(10.0, x);</pre> 
문장: "정렬하라"	연산자: '**'

부프로그램 설계시 고려사항

- 지역변수는 정적 또는 동적으로 할당되는가?
- 부프로그램이 중첩되어 정의 가능한가?
- 어떤 매개변수 전달방법이 제공되는가?
- 매개변수 타입 검사가 이루어지는가?
- 부프로그램이 매개변수로 전달되고 부프로그램이 중첩된다면, 전달된 부프로그램의 참조환경은 무엇인가?
- 부프로그램은 중복될 수 있는가?
- 부프로그램이 포괄형이 될 수 있는가?
- 언어가 중첩 부프로그램을 허용한다면, 클로저는 지원되는가?

지역 참조 환경

- 부프로그램에서 정의된 지역 변수는 지역 참조 환경을 정의
- 지역변수는 정적이거나 스택-동적 (5장 참조)
- 대부분 언어에서 지역변수는 스택-동적

매개변수 전달의 의미적 모델

- 입력 모드(in mode)
- 출력 모드(out mode)
- 입출력 모드 (inout mode)

매개변수 전달의 의미적 모델 (2)

모델	설명
입력 모드	형식 매개변수는 실 매개변수로부터 데이터 전달받음
출력 모드	형식 매개변수는 실 매개변수에 데이터를 전달함
입출력 모드	형식 매개변수는 실 매개변수로부터 데이터를 전달받고, 데이터를 실 매개변수에 전달함

fun(list1, list2) // list1, list2의 배열을 더하여
// 그 결과를 list2로 반환

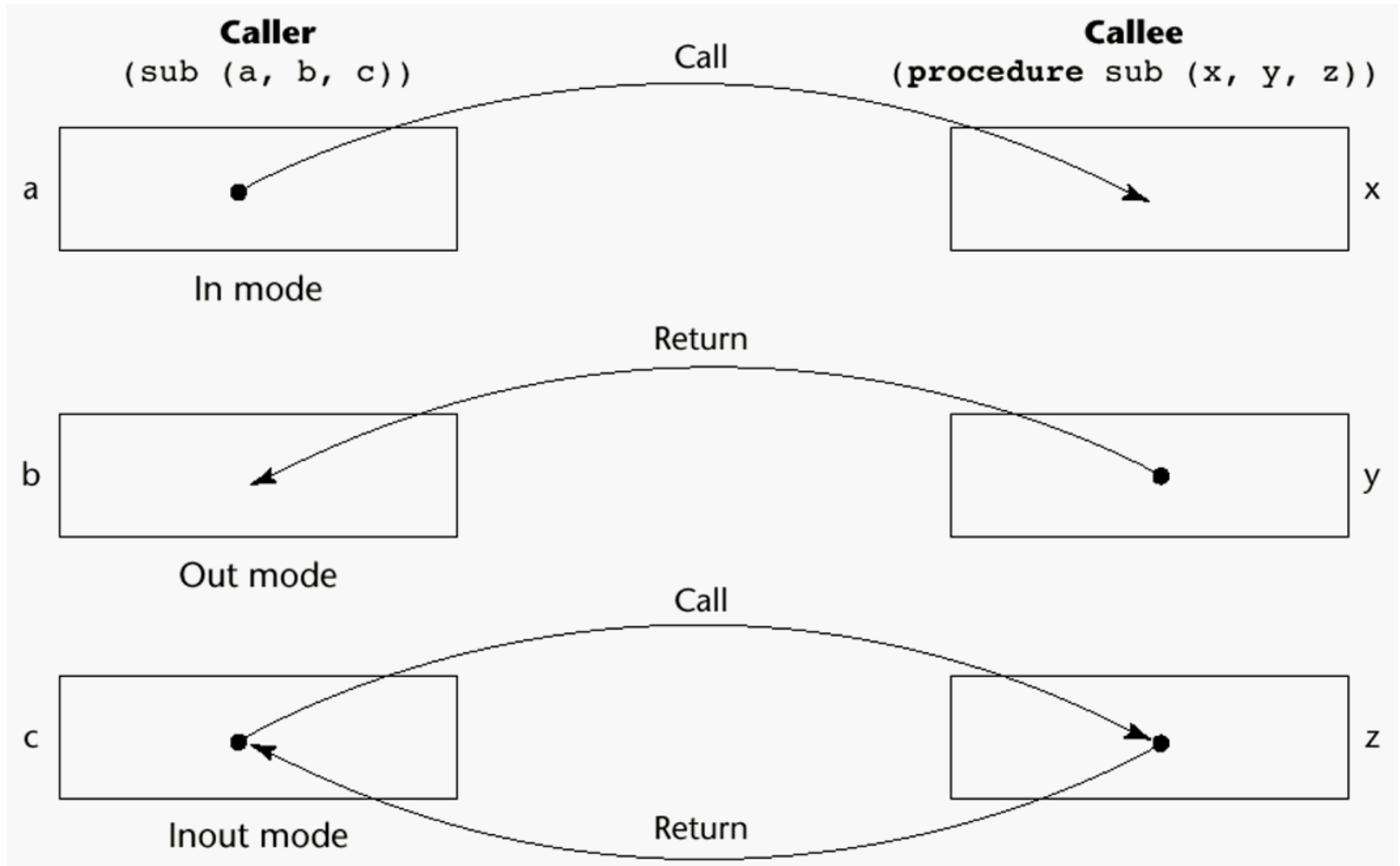
입력 모드

입출력 모드

매개변수 전달의 개념적 모델

- 실제의 값을 전달
- 값의 접근 경로를 전달

매개변수 전달 모델 (값-전달시)



매개변수 전달의 구현 모델

- 값-전달 (pass-by-value)
- 결과-전달 (pass-by-result)
- 값-결과-전달 (pass-by-value-result)
- 참조-전달 (pass-by-reference)
- 이름-전달 (pass-by-name)

값-전달

- **값-전달**(pass-by-value)은 매개변수가 값으로 전달되는 입력 모드의 구현
 - 형식 매개변수는 대응 실 매개변수의 값으로 초기화
 - 형식 매개변수는 지역변수로 사용
 - 데이터 이동은 일반적으로 값 복사로 구현
 - 데이터 이동을 접근 패스 전달로 구현할 경우에 쓰기-보호 셀(읽기만 가능한 셀) 필요

```
int fun(int a, const int &b) // in C++  
{ ... }
```

값-전달 (2)

- 장점:

- 스칼라 변수의 경우 빠른 연결 비용과 접근 시간
- 부프로그램의 외부 데이터 접근 제한

- 단점

- 값 복사가 사용될 경우에, 기억공간 추가 할당, 값 복사
에 따른 비용 부담(배열과 같은 집합체의 경우)
- 접근 패스 전달 경우, 매개변수에 대한 쓰기-보호가 요구
되고, 간접 주소지정에 따른 접근 비용

결과-전달

- 결과-전달(pass-by-result)은 출력 모드의 구현
 - 부프로그램 매개변수에 값이 전달하지 않으며,
 - 형식 매개변수는 지역 변수로 사용되고,
 - 형식 매개변수의 값이 대응 실 매개변수로 전달 (호출 종료시)
 - 데이터 이동은 보통 값 복사로 구현

```
// in C#  
// a, b의 값은 지정될 필요없음
```

```
f.Fixer(out a, out b);
```

```
void Fixer(out int x, out int y)  
// x, y의 값은 반드시 지정되어야 함  
{  
    ...  
    x = 17;  
    y = 35;  
}
```

결과-전달 (2)

- 고려사항

- 실매개변수에 값의 복사 순서는?
- 실 매개변수의 주소가 언제 평가되는가
- Ex. In C#

```
void Fixer(out int x, out int y)
{
    ...
    x = 17;
    y = 35;
}

f.Fixer(out a, out a);
```

```
void Dolt(out int x, int index) {
    x = 17;
    index = 42;
}
...
sub = 21;
f.Dolt(list[sub], sub)
```

sub가 Dolt()
에서 변경되면?

값-결과-전달

- **값-결과-전달**(pass-by-value-result)은 값이 복사되는 입출력 모드의 구현
 - 값-전달과 결과-전달의 혼합
 - **복사-전달**(pass-by-copy)이라고 함

```
int main()
{
    ...
    x = 10; y = 20;
    sub(x, y);
    ...
}
```

```
void sub(int a, int b)
{
    ...
    a = 20; b = 30;
}
```

- 결과-전달, 값-전달의 단점을 공유

참조-전달

- 참조-전달(pass-by-reference)은 입출력 모드의 구현
 - 접근 패스(주로 주소)를 전달
 - 공유-전달(pass-by-sharing)이라고 함
 - Ex. C#

```
int main()
{
    ...
    x = 10; y = 20;
    sub(ref x, ref y);
    ...
}
```

```
void sub(ref int a, ref
int b)
{
    ...
    a = 20; b = 30;
}
```

참조-전달 (2)

- 장점
 - 전달 과정이 효율적
- 단점
 - 형식 매개변수의 느린 접근
 - 원하지 않은 부작용 가능성 (한 방향 전달 요구시)
 - 원하지 않은 별칭 생성

참조-전달 (3)

- 별칭 생성 예: 실 매개변수간의 충돌

```
... fun(total, total)
```

```
... fun(list[i], list[j]) // if i=j?
```

```
... fun(list[i], list)
```

```
void fun(int &first, int &second)  
{ ... }
```


참조-전달 (4)

- 별칭 생성: 형식 매개변수와 가시적 비지역 변수간의 충돌

```
int *global;  
void main() {  
    ...  
    sub(global);  
    ...  
}
```

```
void sub(int *param)  
{ ... }
```

이름-전달 (입출력 모드)

- 실 매개변수는 부프로그램에 나타나는 모든 대응 형식 매개변수를 문자 그대로 대치(textual substitution)
 - 형식 매개변수에 대한 실 매개변수의 값이나 주소에 대한 바인딩은 그 형식 매개변수가 참조될 때까지 지연
 - 구현을 위해서, 매개변수와 함께 호출자의 참조 환경과 매개변수를 평가하는 부프로그램을 전달(부프로그램/참조환경을 클로저라 함)
- 평가
 - 지연 바인딩의 유연성 제공
 - 구현하기 어렵고 비효율적
 - 판독성/신뢰성 저하
 - 포괄적 프로그램의 매개변수 구현시 사용(C++, Java, C#)

예제: 이름-전달

```
...  
a = 2;  
sub(a, a+1, 10);
```

```
void sub(int x, int y, int z)  
{  
    int p;  
  
    p = y;           // p의 값은?  
    x = z;  
    p = y;           // p의 값은?  
}
```

매개변수 전달방법 구현

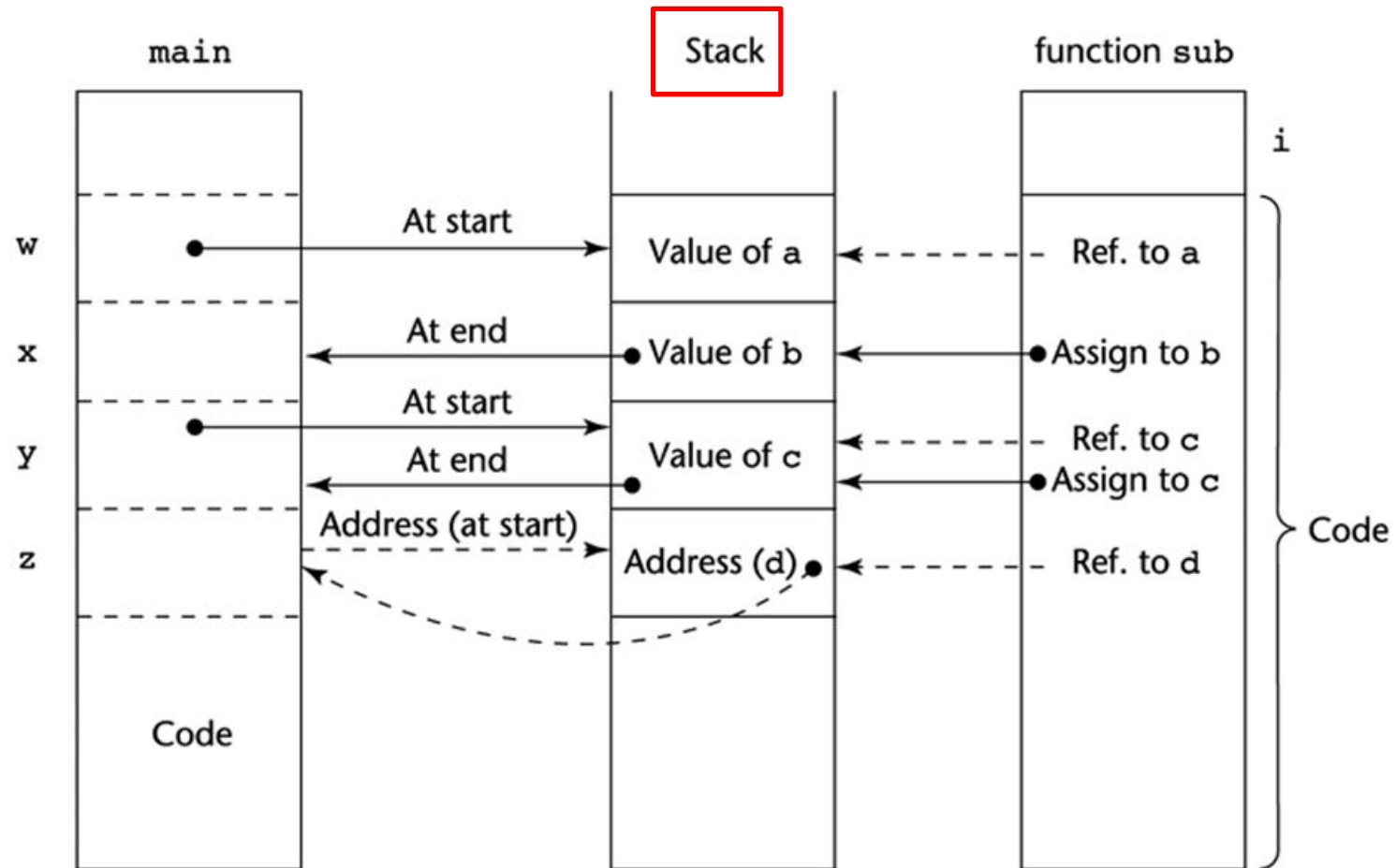
- 대부분의 언어에서, 매개변수 전달은 실행-시간 스택을 통해서 이루어짐
- 참조-전달이 가장 용이함: 단지 주소만 스택에 저장

매개변수 전달방법 구현 (2)

값, 결과, 값-결과, 참조

sub(w, x, y, z)

sub(int a, int b, int c, int d)



주요 언어의 매개변수 전달 방법

언어	매개변수 전달 방법
C	값-전달 참조-전달은?
C++	값-전달 참조 타입을 통한 참조-전달 제공
Java	값-전달 객체 매개변수 전달은?
C#	디폴트는 값-전달 참조-전달: 실, 형식 매개변수 앞에 ref 명세 결과-전달: 실, 형식 매개변수 앞에 out 명세

주요 언어의 매개변수 전달 방법

언어	매개변수 전달 방법
Ada	3가지 의미적 모델 지원: in(디폴트 모드), out, in out in 매개변수는 참조될 수 있으나 배정될 수 없고, Out 매개변수는 배정될 수 있으나 참조될 수 없고, in out 매개변수는 배정, 참조 모두 가능함
Fortran 95	형식 매개변수에 입력, 출력, 입출력 모드 명세 가능
Python	배정-전달(pass-by-assignment): 모든 데이터는 객체이고, 실 매개변수 객체가 형식 매개변수에 배정됨

예제

// in C++

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

// in Ada

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

// in Fortran 95+

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

// in C#

```
void sumer (ref int oldSum, int newOne, out res)
```

...

```
sumer(ref sum, newValue, out res); // sum은 반드시 초기화 필요
```


매개변수의 타입 검사

- 실 매개변수의 타입과 대응 형식 매개변수의 타입과의 일치(consistency) 여부 검사
- 신뢰성을 위해서 매우 중요

매개변수의 타입 검사

- 실 매개변수 float 값을 double 형식 매개변수에 전달하는 함수 호출을 생각해보자.
 - 값-호출로 전달될 때?
 - 참조-호출로 전달될 때?

```
float f = 10.5;
```

```
...
```

```
fun(&f);           // what's the problem in C, C++?
```

```
void fun(double *d) {  
    *d = *d + 10;  
}
```

- C#은 **ref**의 실 매개변수는 대응 형식 매개변수의 타입과 정확히 일치해야 할 것을 요구

주요 언어의 매개변수의 타입 검사

- Fortran 77, 원본 C: 타입 검사하지 않음
- Pascal, Fortran 90, Java, Ada: 항상 요구됨
- C89: 사용자 선택(프로토타입 형식으로 함수 정의시 검사)

// 비 프로토타입 형식 정의

```
double sin(x)
```

```
    double x;
```

```
{    ... }
```

// 프로토타입 형식 정의

```
double sin(double x)
```

```
{    ... }
```

주요 언어의 매개변수의 타입 검사 (2)

- C99, C++
 - 항상 프로토타입 형식 요구됨
 - 매개변수 리스트의 마지막 부분이 생략 기호이면 타입검사를 피할 수 있다.

```
int printf(const char* format_string, ...);
```

```
printf("The sum is %d\n", 3.4);
```

```
printf("The string is %s\n", 10);
```

```
printf("Two integers are %d, %d\n", 10);
```

주요 언어의 매개변수의 타입 검사 (3)

- Perl, JavaScript, PHP: 요구하지 않음
- Python:

매개변수로서의 다차원 배열

- 컴파일러는 배열 원소 참조에 대한 기억장소 사상 함수 구성
- 다차원 배열이 부프로그램에 전달되고, 그 부프로그램이 개별 컴파일가능하면,
 - 컴파일러는 부 프로그램 텍스트만으로 기억장소 사상 함수 구성하는데 필요한 배열 크기 정보 필요
 - 배열의 첫번째 첨자를 제외한 모든 배열 크기 정보 포함 요구
 - 이 경우에 유연한 부프로그램 작성 어려움

$$\begin{aligned}\text{주소}(a[i,j]) &= \text{주소}(a[0,0]) \\ &\quad + ((i\text{번째 행보다 앞선 행의 개수}) * (\text{열의 크기})) \\ &\quad + (j\text{번째 열의 왼쪽에 위치한 원소 개수}) * \text{원소_크기}\end{aligned}$$

매개변수로서의 다차원 배열

- 배열 매개변수에서 열의 크기만 다를 경우에 별도의 함수를 작성해야 하는가?

```
void fun(int matrix[][10])  
{ ...}  
void fun(int matrix[][20])  
{ ...}
```

매개변수로서의 다차원 배열 (2)

- In C/C++

- 해결책은 행렬을 포인터로, 행렬 크기를 매개변수로 전달
- 사용자가 행렬 크기 매개변수를 이용하여 기억장소 사상 함수 표현

```
#define mat_ptr(r,c) (*mat_ptr + ((r)* (num_cols) + (c))  
void fun(int *mat_ptr, int num_rows, int num_cols)  
{  
    // mat_ptr[i][j]의 표현  
    // *(mat_ptr + (i * num_cols) + col)  
    mat_ptr(i, j) = x;  
    ...  
}
```


매개변수로서의 다차원 배열 (3)

- Java/C#

- 배열은 객체
- 배열은 모두 1차원이고, 그러나 원소는 배열 가능
- 각 배열은 객체 생성시에 그 길이로 설정 length(C#에서는 Length) 포함

```
float sumer(float mat[][]) {  
    float sum = 0.0f;  
    for (int row = 0; row < mat.length; row++) {  
        for (int col = 0; col < mat[row].length; col++) {  
            sum += mat[row][col];  
        } ...  
    }
```

매개변수 전달 고려사항

- 2가지 중요한 고려사항
 - 효율성
 - 데이터 이동에서 단방향과 양방향의 선택
- 위에서 선택 기준은 충돌
 - 좋은 프로그래밍은 변수의 제한된 접근 제안 (가능하면 단 방향을 선택)
 - 그러나 크기가 큰 집합체 데이터는 참조-호출이 효율적