

프로그래밍 언어론

Subprograms

컴퓨터공학과
이만호



충남대학교
CHUNGNAM NATIONAL UNIVERSITY

Subprograms

학 습 목 표

Subprogram을 구체적으로 학습하기에 필요한
기초적이고도 전반적인 내용을 학습한다.

학 습 내 용

- Abstraction(추상화)
- Subprogram 일반
- 지역 참조환경



목 차

- 알고가기
- 학습하기
 - Abstraction(추상화)의 기능
 - Subprogram의 필요성
 - Subprogram 관련 용어 정의
 - Parameter 대응
 - Subprogram 설계 시 고려 사항
 - 지역 참조환경
- 평가하기
- 정리하기



알고가기



아래에서 설명하고 있는 것 중에서 “간단히 표현한다”는 특성이 다른 것과 다른 것은?

- (a) “ $a + b$ ”는 변수 a 와 b 의 값을 더하는 일을 수행하는 여러 개의 assembly 언어 명령을 간단히 표현한 것이다.
- (b) “ $n!$ ”은 1부터 n 까지의 모든 정수를 곱하는 일을 간단히 표현한 것이다.
- (c) “ $\text{sort}(n, \text{score})$ ”은 n 개의 요소를 가진 배열 score 를 정렬하는 일을 수행하는 여러 개의 문장을 간단히 표현한 것이다.
- (d) 변수(variable)라고 하는 것은 이름이 영문자로 시작하며 기억공간을 차지할 수 있는 객체 모두를 간단히 표현한 것이다.

확인



| Abstraction(추상화)의 기능

Process Abstraction

- 어떤 목적을 수행하는 일련의 문장들을 subprogram으로 작성함
- Subprogram의 종류
 - Procedure
 - Function(함수)
- Readability, writability 향상
- 프로그래밍 초기부터 강조되어 왔음

Data Abstraction

- 프로그래머가 새로운 data type을 정의하는 것을 허용함
- 1980년대부터 강조됨
- 객체지향언어에서 다루게 됨



| Subprogram의 필요성

⇒ 아래의 프로그램을 고려

```

...
A  distx = x1 - x2;
   if (distx < 0) distx = -distx;
...

```

```

B  disty = y1 - y2;
   if (disty < 0) disty = -disty;
...

```

>> 두 부분이 유사함: 변수 이름을 제외하면 동일

⇒ 위의 유사한 부분과 비슷하지만 다르게 코드를 작성

```

C  dist = s - t;
   if (dist < 0) dist = -dist;

```

>> (A의 x1)과 (B의 y1)은 (C의 s)에 대응

>> (A의 x2)와 (B의 y2)는 (C의 t)에 대응

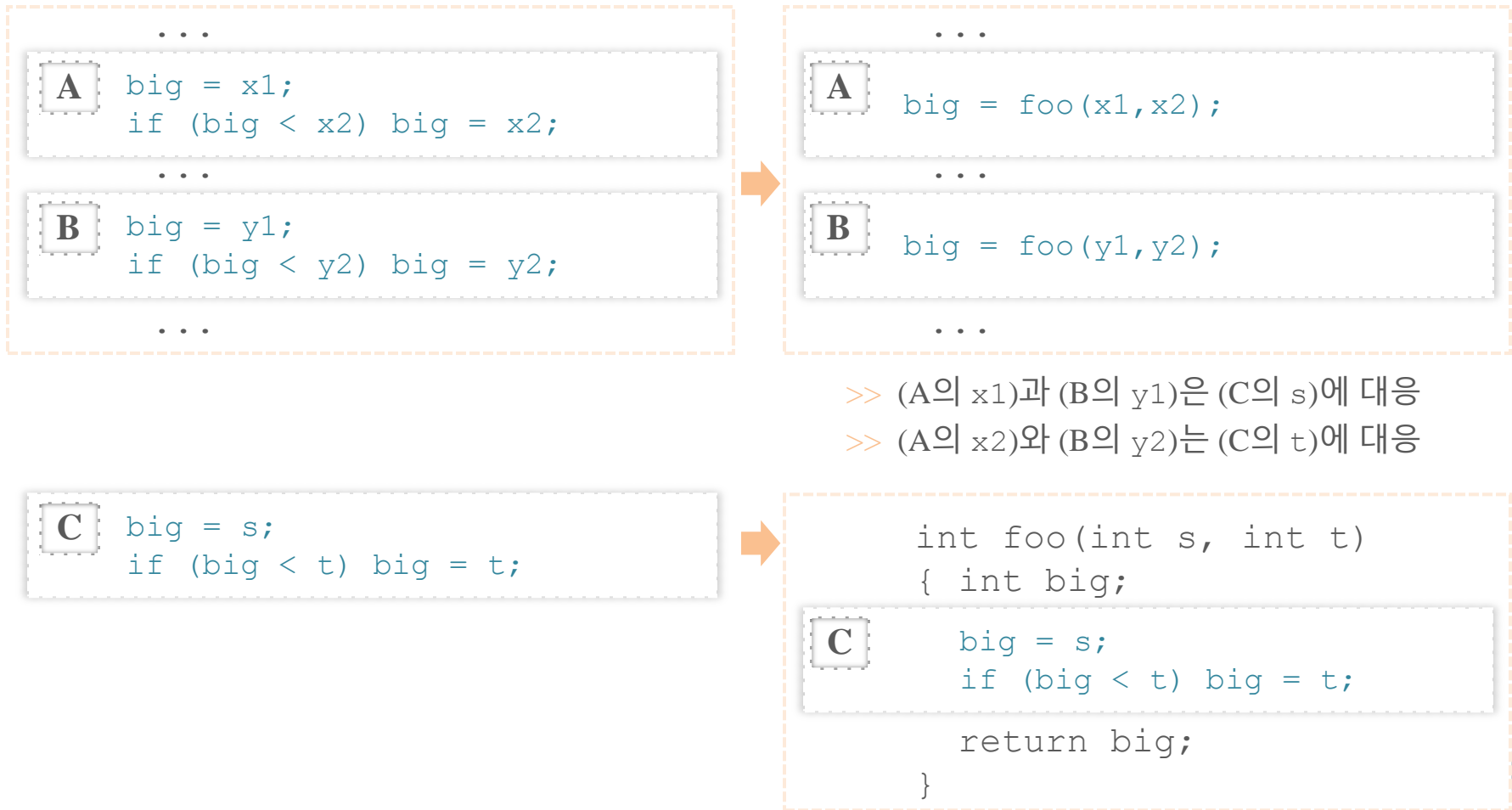
⇒ C 부분을 subprogram으로 작성하여 A와 B부분에서 활용하는 방법이 있으면 편리하겠음

→ 대부분의 언어에서 지원



| C 언어의 Subprogram 구현

➡ Subprogram을 사용



| Subprogram 사용 이유

반복되는 코드를 한번만 작성함으로써 코드 작성이 용이

프로그램 module화가 용이

- 큰 문제는 작은 문제들로 나누어 해결하는 것이 일반적인 문제 해결 접근 방법임
 - 작은 문제 각각은 subprogram(즉, module)으로 구현됨
- 한 프로그램을 작은 subprogram들의 집합으로 정의함
- 각 subprogram은 독립적으로 작성되고, 수정되고, 테스트됨

개별 컴파일(separate compilation)이 가능

- Subprogram들은 대개 서로 독립적임
 - 각 subprogram은 독립적으로 컴파일될 수 있음
 - Compile된 subprogram들은 linking을 통해서 실행 가능한 큰 프로그램이 됨
- 한 subprogram이 수정되면, 해당 subprogram만 다시 compile하면 됨
 - 한 subprogram의 수정으로 프로그램 전체를 다시 compile할 필요가 없음
 - 규모가 큰 프로그램의 경우 compile 시간을 상당히 절약할 수 있음



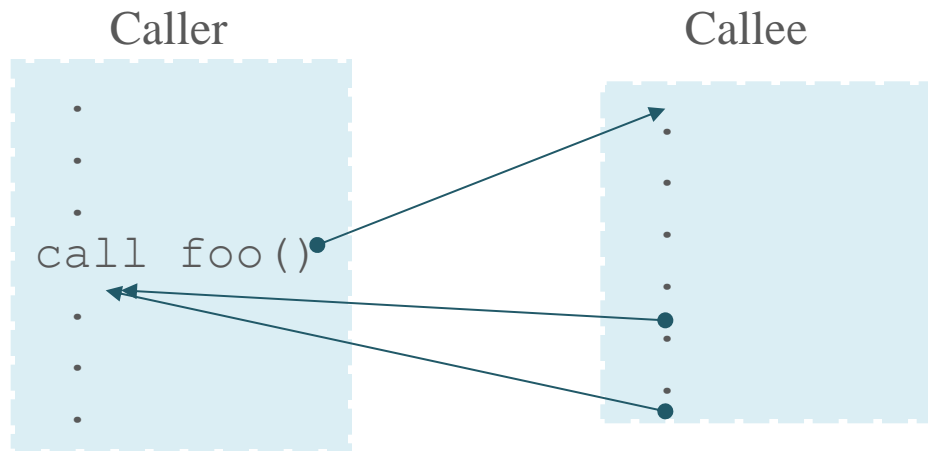
| Subprogram의 특성

한 subprogram의 실행이 시작되는 지점은 하나임 (one-entry)

한 subprogram의 실행이 종료되는 지점은 하나 이상일 수 있음

Subprogram의 Caller와 Callee의 동작 과정

- Caller(subprogram을 호출한 module)는 callee(호출된 subprogram)의 실행이 완료될 때까지 실행이 유보됨
- Callee의 실행이 완료되면 프로그램의 실행제어는 caller에게로 넘어감
- Caller는 subprogram을 호출한 지점의 다음부터 실행을 재개함



| Subprogram 관련 용어 정의 - 1

Subprogram의 종류

- 함수(function), Procedure

Subprogram 이름

- 다른 subprogram과 구별하기 위한 이름

Formal parameter(형식 매개변수)

- Subprogram header에 나열된 임시변수(dummy variable)
- Subprogram 내에서만 사용됨

Parameter profile

- Signature라고도 함
- Formal parameter의 개수(number), 순서(order), type

Return type

- Subprogram이 함수(function)인 경우에만 사용됨
- Function을 실행한 결과로 return하는 값의 type

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

    ...
    bigger = foo(a, b);
    ...
}

int foo(int s, int t)
{
    int big;

    big = s;
    if (big < t)
        big = t;
    return big;
}
```

| Subprogram 관련 용어 정의 - 2

Subprogram header

- Subprogram의 첫 부분으로 subprogram의 interface(사용법)를 표현
- Subprogram의 이름(name), 종류, parameter profile, return type(함수의 경우)

Subprogram body

- Subprogram의 action(실행과정)을 표현하는 부분

Protocol

- Procedure의 "parameter profile"과 동일
- Function(함수)인 경우에는 return type이 추가됨.
즉, {parameter profile, return type}

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

    ...
    bigger = foo(a, b);
    ...
}

int foo(int s, int t)
{
    int big;

    big = s;
    if (big < t)
        big = t;
    return big;
}
```



| Subprogram 관련 용어 정의 - 3

Subprogram 정의(definition)

- ➔ Subprogram의 interface(사용법)와 action(실행 과정)을 표현

Subprogram 선언(declaration)

- ➔ Subprogram의 header 부분, body는 포함하지 않음
- ➔ C 언어에서는 prototype이라고 함

Subprogram 호출(call)

- ➔ Subprogram을 실행하도록 하는 명시적 표현
- ➔ Subprogram 이름과 actual parameter

Actual parameter(실 매개변수)

- ➔ Subprogram을 호출할 때 사용되는 값(value)이나 주소(address)
- ➔ Formal parameter에 대응됨

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

    ...
    bigger = foo(a, b);
    ...
}

int foo(int s, int t)
{
    int big;

    big = s;
    if (big < t)
        big = t;
    return big;
}
```

| Parameter(매개변수)의 대응

Actual parameter와 Formal parameter를 대응

대응 방법

Positional parameter(위치 매개변수)

- ⇒ Parameter의 순서에 따라 대응
- ⇒ 지원하는 언어: 대부분의 언어에서 지원함

예 `sort(int row, int col, int array2) { ... }`
`... sort(50, 7, score); ...`

Keyword parameter

- ⇒ Actual parameter에 대응될 formal parameter 이름을 actual parameter와 함께 지정
- 장점 Parameter의 순서를 기억할 필요 없음
- 단점 Formal parameter 이름을 정확하게 알아야 함
- ⇒ 지원하는 언어: Ada, Python

예 `... sort(array2=>score, row=>50, col=>7); ...`

혼합 형태

- ⇒ Positional parameter가 앞에 사용되고, keyword parameter는 뒤에 나타남
- ⇒ 지원하는 언어: Ada, Python

예 `... sort(50, array2=>score, col=>7); ...`



| Default Parameter

- Formal parameter로 actual parameter가 전달되지 않을 경우, formal parameter가 가지는 값을 지정할 수 있음
- 지원하는 언어: Ada, Python, C++, PHP, Ruby

예

Python

- Default parameter 이후의 actual parameter는 keyword parameter이어야 함

```
def compute_pay(income, exemptions=1, tax_rate)
... pay=compute_pay(5000.0, tax_rate=0.15) ...
```

예

C++

- Default parameter는 맨 마지막에 사용됨

```
float compute_pay(float income, float tax_rate,
                  int exemption=1)
... pay=compute_pay(5000.0, 0.15) ...
```



| 가변 개수의 Parameter - 1

➔ 다양한 개수의 actual parameter가 전달될 수 있음

예

C#

➔ 가변 formal parameter에 “params” 라는 keyword가 있어야 함

```
class MyClass {  
    ...  
    public void DisplayList(params int[] list) {...}  
}  
MyClass myObj = new MyClass;  
int[] myList = new int[6] {2,4,6,8,10,12};  
myObj.DisplayList(myList);           // pass array  
myObj.DisplayList(1,3,5,7,9);        // pass list
```



| 가변 개수의 Parameter - 2

예

Python

- Formal parameter는 3가지가 순서대로 사용될 수 있음
 - > Positional parameter
 - > 상수 배열(constant array): Python에서는 tuple이라고 부름
 - >> Formal parameter 이름 앞에 '*'를 붙임
 - > Hash: Python에서는 dictionary라고 부름
 - >> Formal parameter 이름 앞에 "**"를 붙임
 - >> 여러 개의 "key=value" 형태가 actual parameter로 전달됨

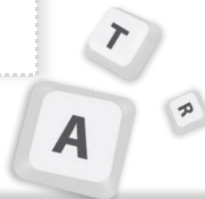
```
def foo(p1, p2, *a, **h); ...
... foo(2, 4, 6, 8, mon=68, tue=72, wed=77) ...
    p1:2, p2:4, a:[6,8],
    h: {'mon'=68, 'tue'=72, 'wed'=77}
```

기타 예

Ruby

Lua

C의 printf(...)



| Subprogram의 종류

Subprogram의 2가지 종류

➔ Procedure

➔ Function(함수)

Procedure	Function(함수)
Parameter를 포함하는 문장들의 집합을 추상화	수학적 함수를 모델링
사용자가 새 문장 을 정의	사용자가 새 연산자(operator) 를 정의
실행 결과는 비지역 변수나 parameter를 이용하여 caller에게 전달	실행 결과로 값 한 개를 생성하여 caller에게 전달
실행 결과 전달을 위해 side-effect(부수효과)를 이용	Side-effect가 없다고 생각할 수 있으나, side-effect가 발생하는 경우도 있음
<pre>void sort(int list[], int listlen); ... sort(scores, 100);</pre>	<pre>float power(float base, float exp); ... result = 3.4 * power(2.0, x);</pre>



| Subprogram 설계 시 고려 사항

- ➔ 지역변수 할당이 static(정적)인가 또는 dynamic(동적)인가?
- ➔ Subprogram이 다른 subprogram 안에 정의될 수 있는가?
(즉, subprogram이 중첩(nested)될 수 있는가?)
- ➔ 어떤 parameter 전달 방법이 사용되는가?
- ➔ Actual parameter가 formal parameter로 전달될 때 type check가 이루어지는가?
- ➔ Subprogram이 parameter로 전달되고 subprogram이 중첩(nested)된다면, 전달된 subprogram의 참조 환경(referencing environment)은 무엇인가?
- ➔ Subprogram은 overload(중복)될 수 있는가?
- ➔ Subprogram은 generic(포괄형)이 될 수 있는가?



| 지역 참조환경

지역 참조환경

- Subprogram은 지역변수로 구성되는 지역 참조환경(local referencing environment)을 정의한다.
- 지역변수는 static 변수 또는 stack-dynamic 변수이다.

지역변수가 stack-dynamic 변수인 경우

장점

- Recursion 지원 가능
- 지역변수에 대한 기억공간이 다른 subprogram들과 공유 가능

단점

- Subprogram이 실행될 때마다, 지역변수 할당/반납/초기화 시간이 필요함
- Indirect-addressing(간접-주소) 방식에 의한 지역변수 접근 시간이 길다.
- Subprogram은 history-sensitive(과거-민감)가 될 수 없다.

지역변수가 static 변수인 경우

- 장점과 단점이 stack-dynamic경우의 반대

```
int rand(int lim)
{ long          d = 2796203;
  static long a = 100001;

  a = (a * 125) % d;
  return ((a % lim) + 1); }
```

평가하기

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

START



평가하기 1

1. Subprogram을 사용하는 이유로 적절하지 않은 것은?

- (a) 반복되는 코드를 단지 한번만 작성할 수 있기 때문에
- (b) 프로그램 실행 시간을 줄일 수 있기 때문에
- (c) 큰 문제는 작은 문제들로 나누어 해결하기가 용이하므로
- (d) 프로그램 개발 시 컴파일 시간을 줄일 수 있기 때문에

확인



평가하기 2

2. Subprogram에 대한 설명으로 옳바르지 않은 것은?

- (a) Subprogram의 한 종류인 procedure는 side-effect를 이용하여 실행 결과를 전달한다.
- (b) C++에서 default parameter는 항상 마지막에 위치해야 한다.
- (c) Subprogram의 지역 변수가 모두 static이면, recursion 지원이 불가능하다.
- (d) Subprogram의 한 유형인 function은 operator의 확장 개념으로 생각할 수 있다.

확인



정리하기

➡ Subprogram의 종류

- Procedure
- Function(함수)

➡ Subprogram 관련 용어

- Subprogram 이름(subprogram name)
- Formal/Actual parameter(형식/실 매개변수)
- Parameter profile
- Return type
- Subprogram header
- Subprogram body
- Protocol
- Subprogram 정의(definition)
- Subprogram 선언(declaration)
- Subprogram 호출(call)



“ 강의를 마치겠습니다. 수고하셨습니다. ”

