

프로그래밍 언어론

10 Simple Subprogram의 구현 (Implementing Simple Subprogram)

컴퓨터공학과
이만호



Simple Subprogram 의 구현

학 습 목 표

- Nested subprogram을 허용하지 않는 언어에서 subprogram의 구현 방법에 대해서 학습한다.

학 습 내 용

- Subprogram의 호출과 복귀의 의미
- Simple Subprogram의 구현
- Run-Time Stack(실행시간스택)을 이용한 Subprogram의 구현



목 차

- 들어가기
- 학습하기
 - Subprogram 호출과 복귀의 의미
 - Simple Subprogram 구현하기
 - Run-time stack을 이용한 Subprogram의 구현
- 평가하기
- 정리하기



알고가기 1



Subprogram을 호출하는 호출자(caller)가 호출될 subprogram에게 전달할 수 있는 것이 아닌 것은?

- a. Actual parameter의 값(value)/주소(address)
- b. Actual parameter의 type
- c. Return address(복귀주소)
- d. 호출될 subprogram의 지역변수(local variables)

확인



| Subprogram 호출(Call)과 Return(복귀)의 의미

Subprogram의 Call과 Return 작업 : Subprogram linkage(연결)라고 부른다.

Subprogram을 Call한다는 것의 의미

- ▶ Parameter passing(Parameter 전달)
- ▶ 지역변수(local variable)가 사용할 기억장소 할당
- ▶ 호출자(caller)의 실행 상태 저장 (register, CPU 상태, 실행 환경 등)
- ▶ Return(복귀)할 수 있는 방법을 알려 주어야 함
- ▶ 비지역변수(non-local variable)에 접근할 수 있는 방법을 알려 주어야 함
- ▶ 제어(control)를 subprogram 진입 지점(entry point)으로 넘김

Subprogram에서 Return한다는 것의 의미

- ▶ Out mode 또는 Inout mode의 parameter 값을 actual parameter에 전달
- ▶ 지역변수에 할당된 기억장소를 반납
- ▶ 호출자(caller)의 실행 상태를 복원(restore)
- ▶ 제어를 호출자의 return(복귀) 지점으로 넘김
- ▶ Subprogram이 함수이면, 호출자가 받을 수 있는 공간에 결과 값을 전달



| Simple Subprogram 구현하기

Simple Subprogram이란

- Subprogram이 중첩되지 않음(not nested)
- 모든 지역변수는 static(정적)임
- 초기의 Fortran

Simple Subprogram을 호출하기

- 호출자의 실행상태를 저장 (기억 공간 필요)
- Actual parameter를 Formal parameter로 전달 (기억 공간 필요)
- Return할 주소를 피호출자(callee)에게 전달 (기억 공간 필요)
- 제어를 피호출자(callee)에게 넘김

Simple Subprogram에서 Return하기

- Parameter가 out mode나 inout mode이면, 그 값을 actual parameter로 전달
- Subprogram이 함수이면, 함수 값을 호출자(caller)에게 전달 (기억 공간 필요)
- 호출자(caller)의 실행상태를 복원
- 제어를 호출자(caller)에게 넘김



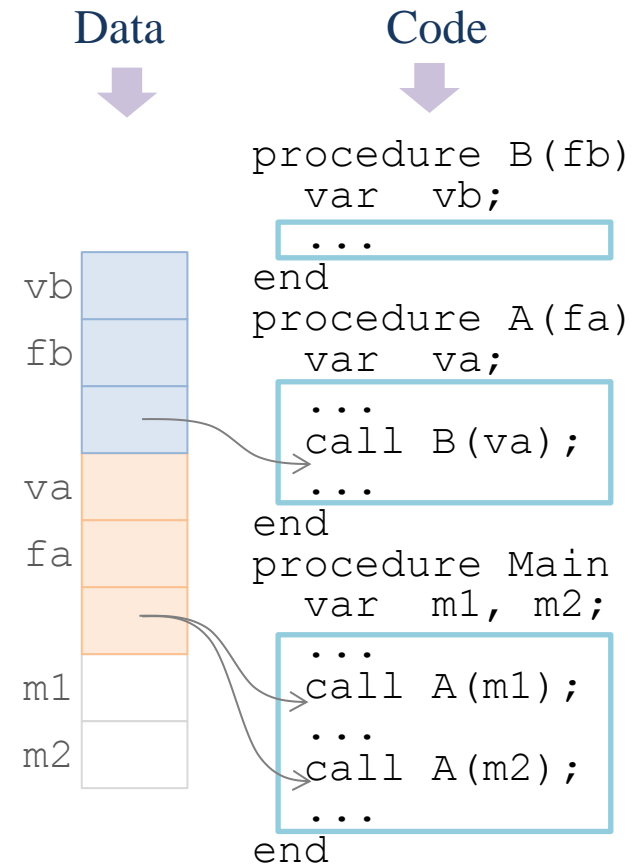
| Simple Subprogram의 구성 요소

Code 부분

- ➔ Subprogram의 작업을 실행
- ➔ Subprogram 실행 과정에서 변하지 않음

Data 부분(non-code 부분)

- ➔ Formal parameter, 지역변수, Return address
- ➔ Subprogram 실행 과정에서 값이 변함
- ➔ 각 subprogram마다 하나씩 static하게 생성됨
 - ▶ Non-code 부분의 형식(format 또는 layout)을 **Activation Record(활성레코드, AR)**라고 함
 - ▶ **Activation Record** 형식에 따라 실제 data가 저장되는 기억장소를 **Activation Record Instance(활성레코드 사례, ARI)**라고 함
 - ▶ ARI의 크기는 지역변수 개수 및 parameter 개수에 따라 달라짐



AR:

Local variables
Parameters
Return address



| Run-Time Stack을 이용한 Subprogram의 구현

Subprogram이 호출될 때마다 ARI를 Run-time stack(실행시간스택, RTStack)에 할당

- ➔ ARI를 RTStack에 할당해 주고, 필요한 정보를 ARI에 저장해 주는 code를 compiler가 생성해 주어야 한다.
- ➔ Subprogram의 Recursive call(재귀호출)이 가능해진다.
 - ▶ Subprogram code는 하나
 - ▶ Subprogram code가 실행되면서 사용하는 ARI는 여러 개

AR(활성레코드)에 필요한 정보

- ➔ 지역변수(local variables)
- ➔ Formal parameters
- ➔ **Static link(정적링크)**
- ➔ **Dynamic link(동적링크)**
- ➔ Return address(복귀주소)

AR:

Local variables
Parameters
Static link
Dynamic link
Return address

※ Static link는 block 구조를 지원하는 언어에서만 필요



| Run-Time Stack(RTStack)의 구조

Top pointer

➔ RTStack의 가장 꼭대기 주소

Environment Pointer(환경포인터, EP)

➔ 현재 실행 중인 subprogram이 사용하고 있는 ARI의 밑바닥 주소

Static Link(정적링크)

➔ 해당 ARI를 사용하고 있는 subprogram의 static parent(정적부모)가 사용하고 있는 ARI의 밑바닥 주소

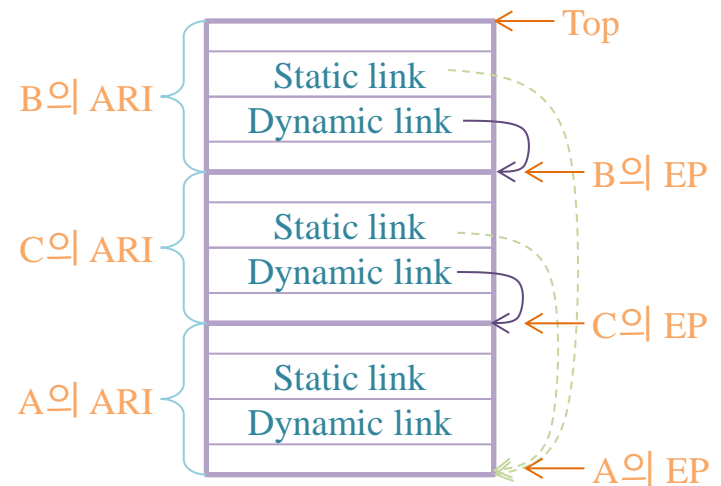
Dynamic Link(동적링크)

➔ 해당 ARI를 사용하고 있는 subprogram을 호출한 subprogram이 사용하고 있는 ARI의 꼭대기 주소

➔ Dynamic chain(동적체인, 호출체인, call chain)을 구성한다.

```

proc A(...)
  ...
  proc B(...)
    ...
  end
  proc C(...)
    ...
    call B(...);
    ...
  end
  ...
  call C(...);
  ...
end
  
```

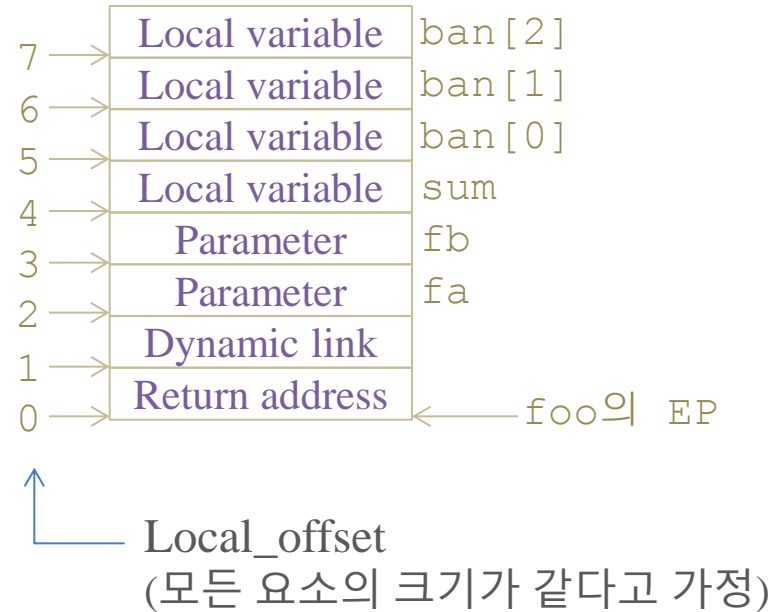


| C 함수의 ARI

```
void foo(int fa, int fb)
{
    int sum, ban[3];
    ...
}
```

함수 `foo`가 호출되면
그림과 같은 ARI가
RTStack에 생성된다.

foo의 ARI



C 언어는 block 구조를 지원하지 않으므로 Static link는 필요 없다.



| Recursive Call이 없는 경우의 RTStack

```

void fun1(int x) {
    int y;
    ...
    fun3(y);
    ...
}

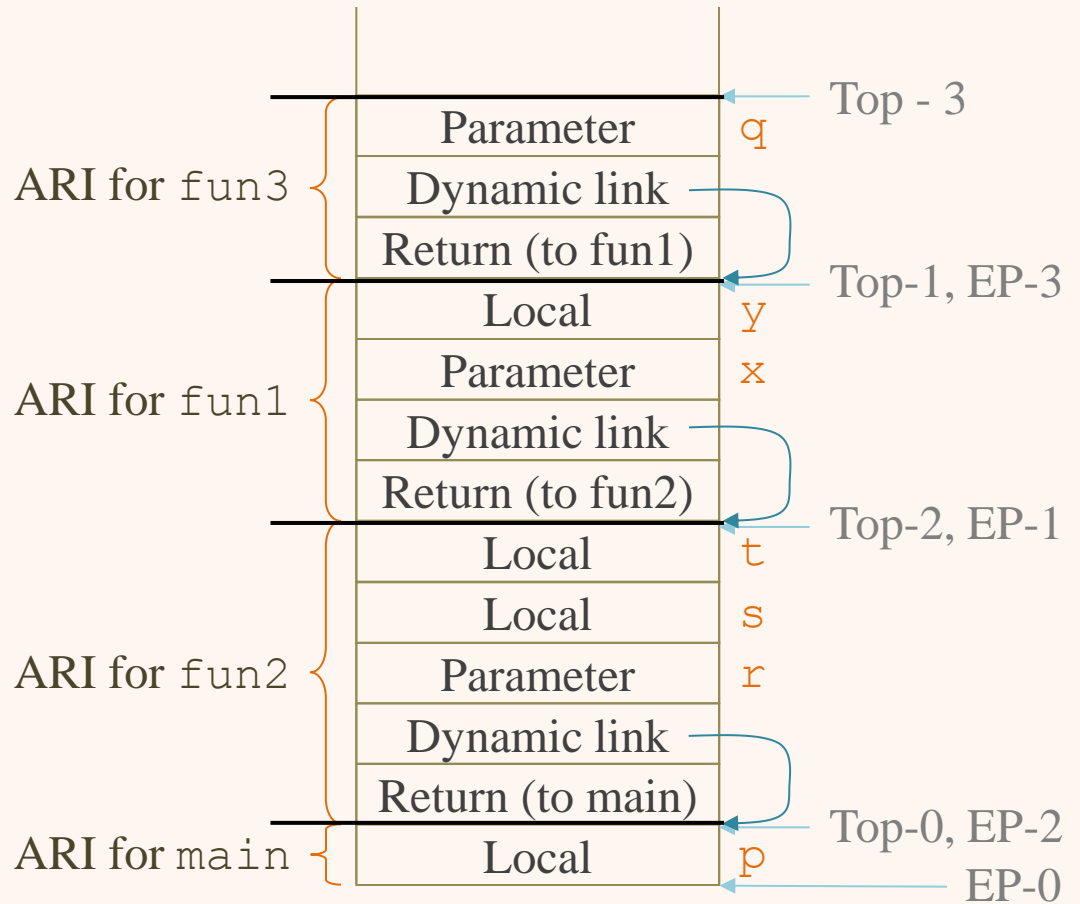
void fun2(float r) {
    int s, t;
    ...
    fun1(s);
    ...
}

void fun3(int q) {
    ...
}

void main() {
    float p;
    ...
    fun2(p);
    ...
}

```

Arrows in the original image point to the recursive calls: fun3(y) in fun1, fun1(s) in fun2, and fun2(p) in main.

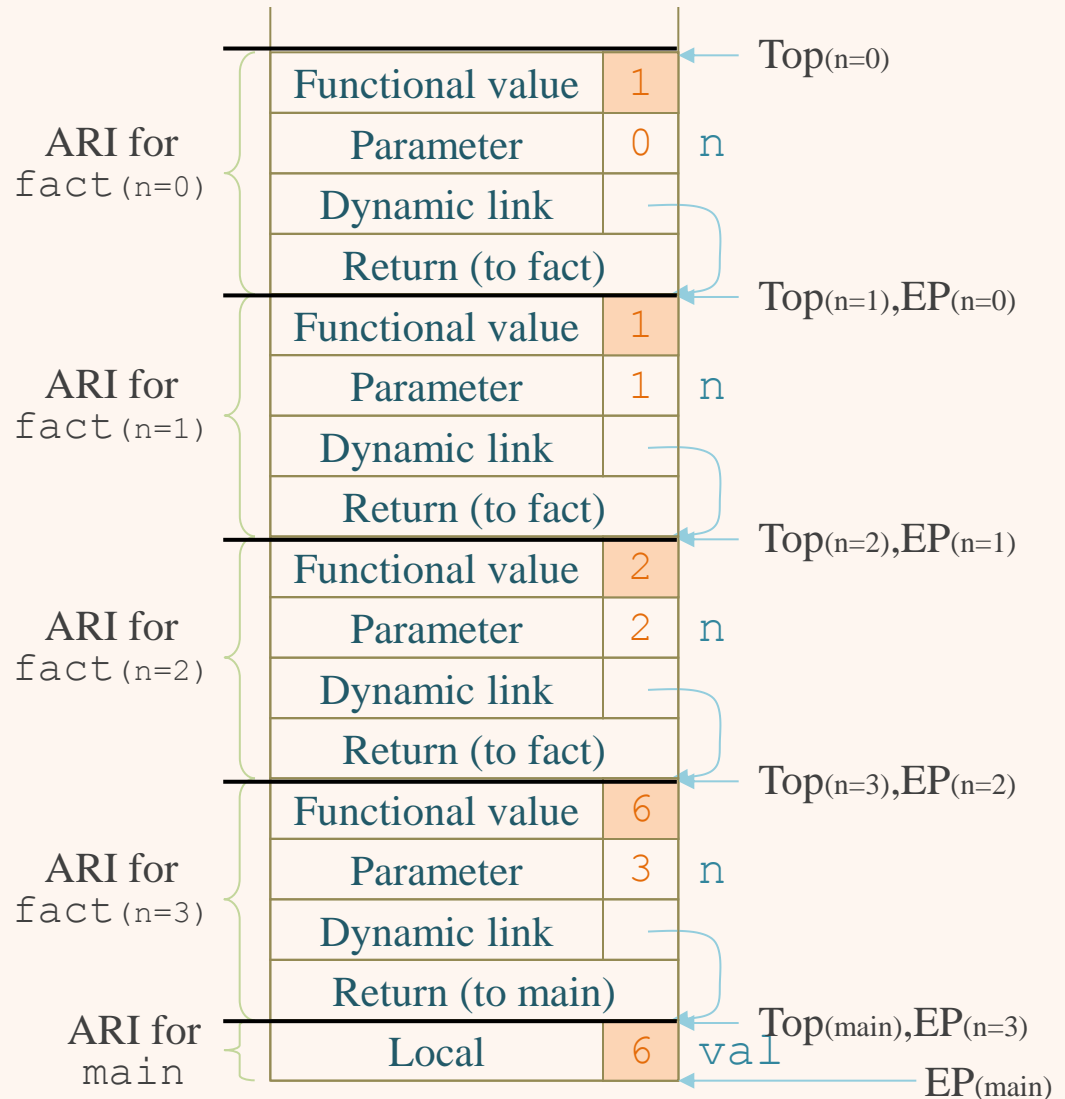


| Recursive Call이 있는 경우의 RTStack

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n * (n-1)! & \end{cases}$$

```
int fact(int n) {
    <-----1
    if (n == 0)
        return 1;
    else
        return n*fact(n-1);
    <-----2
}
void main() {
    int val;
    val = fact(3);
    <-----3
}
```

함수는 결과값 하나를 return해야 하므로 ARI에 결과값을 위한 장소 하나가 추가된다.



평가하기

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

START



평가하기 1

1. Subprogram의 중첩(nesting)을 허용하지 않는 언어에서, subprogram의 activation record(활성레코드, AR)에 포함되지 않는 것은?
- a. Subprogram의 지역변수 (local variables)
 - b. Subprogram의 formal parameters
 - c. Static link(정적링크)
 - d. 호출한 subprogram으로의 return address(복귀주소)

확인



평가하기 2

2. Activation record instance(활성레코드사레, ARI)에 관한 설명으로 맞지 않는 것은?

- a. 한 subprogram이 실행될 때마다 생성되는 ARI의 형식은 항상 같다.
- b. 한 subprogram이 실행될 때마다 생성되는 ARI의 내용은 다를 수 있다.
- c. Procedure에 대응되는 ARI의 구성요소와 함수에 대응되는 ARI의 구성요소는 같다.
- d. 한 subprogram에 대해서 한 시점에 여러 개의 ARI가 존재할 수 있다.

확인



정리하기

➡ Subprogram 호출과 Return의 의미

- ▶ Subprogram을 호출할 때 이루어지는 작업
- ▶ Subprogram에서 복귀할 때 이루어지는 작업

➡ Simple Subprogram

- ▶ Subprogram 중첩(nested)이 허용되지 않음
- ▶ 모든 지역변수는 Static(정적)임

➡ Activation Record(AR)와 Activation Record Instance(ARI)

- ▶ AR은 형식(format)
- ▶ ARI는 subprogram이 실행될 때 AR 형식으로 생성됨

➡ Top과 Environment Pointer(EP)

- ▶ Top: Run-Time Stack의 꼭대기 주소
- ▶ EP: Subprogram이 실행될 때 사용하는 ARI의 밑바닥 주소

➡ Dynamic link(동적링크)

- ▶ ARI를 사용하는 subprogram의 호출자의 ARI의 꼭대기 주소



“ 강의를 마치겠습니다. 수고하셨습니다. ”

