

# 10장 부프로그램 구현

2020. 6. 29

순천향대학교 컴퓨터공학과

# 목차

- 부프로그램의 호출과 복귀의 의미
- 단순 부프로그램의 구현
- 스택-동적 지역변수를 갖는 부프로그램의 구현
- 중첩 부프로그램 구현

# 부프로그램 호출과 복귀 의미

- 부프로그램 호출과 복귀 연산을 **부프로그램 연결** (subprogram linkage)라 한다.
- 부프로그램 연결 의미에 기반하여 부프로그램 구현

# 부프로그램 호출과 복귀 의미

- 부프로그램 호출 연산 의미

- 매개변수 전달 방법 구현
- 피호출자의 스택-동적 지역변수에 대한 기억공간 할당
- 호출자의 실행 상태 저장(레지스터 값, CPU 상태, 환경 포인터)
- 복귀 주소를 피호출자에게 전달 (부프로그램 복귀 마련)
- 제어를 피호출자에게 전달
- 중첩 부프로그램 지원시, 피호출자에 가시적인 비지역변수 접근 방법 마련

# 부프로그램 호출과 복귀 의미

- 부프로그램 복귀 연산 의미

- 매개변수가 출력 또는 입출력 모드이고, 복사-전달로 구현되는 경우 형식 매개변수의 값을 대응 실 매개변수로 전달
- 스택-동적 지역변수에 할당된 기억공간 해제
- 호출자의 실행 상태를 복원
- 제어를 호출자에게 반환

# 단순 부프로그램 구현

- 단순 부프로그램이란?
  - 부프로그램은 중첩되지 않는다.
  - 모든 지역변수는 정적이다.
- 호출 의미:
  - 호출자의 실행 상태 저장
  - 매개변수 전달
  - 복귀 주소 전달
  - 피호출자에게 제어 전달

# 단순 부프로그램 구현 (2)

- 복귀 의미:

- 매개변수가 출력 모드 또는 값-결과-전달이면 형식 매개변수의 값을 대응 실 매개변수로 전달
- 함수일 경우에, 함수 값을 호출자에게 전달
- 호출자의 실행 상태 복원
- 제어를 호출자에게 전달

- 호출/복귀에 필요한 기억공간

- 상태 정보, 매개변수, 복귀 주소, 함수 반환 값, 임시 기억공간

## 단순 부프로그램 구현 (2)

- 단순 부프로그램의 2가지 구성 부분
  - 부프로그램 코드(변하지 않는 부분)
  - 비 코드 부분(변하는 부분: 지역변수, 매개변수, 복귀주소)
  - 단순 부프로그램에서 위의 2가지 부분은 고정된 크기임
- 부프로그램의 비 코드 부분의 형식(format, layout)을 **활성화 레코드**(activation record, AR)라 함
- **활성화 레코드 사례**(activation record instance, ARI)는 특정 활성화된 부프로그램의 활성화 레코드의 실제 예이며, AR에 존재하는 데이터 모임.



## 단순 부프로그램 구현 (2)

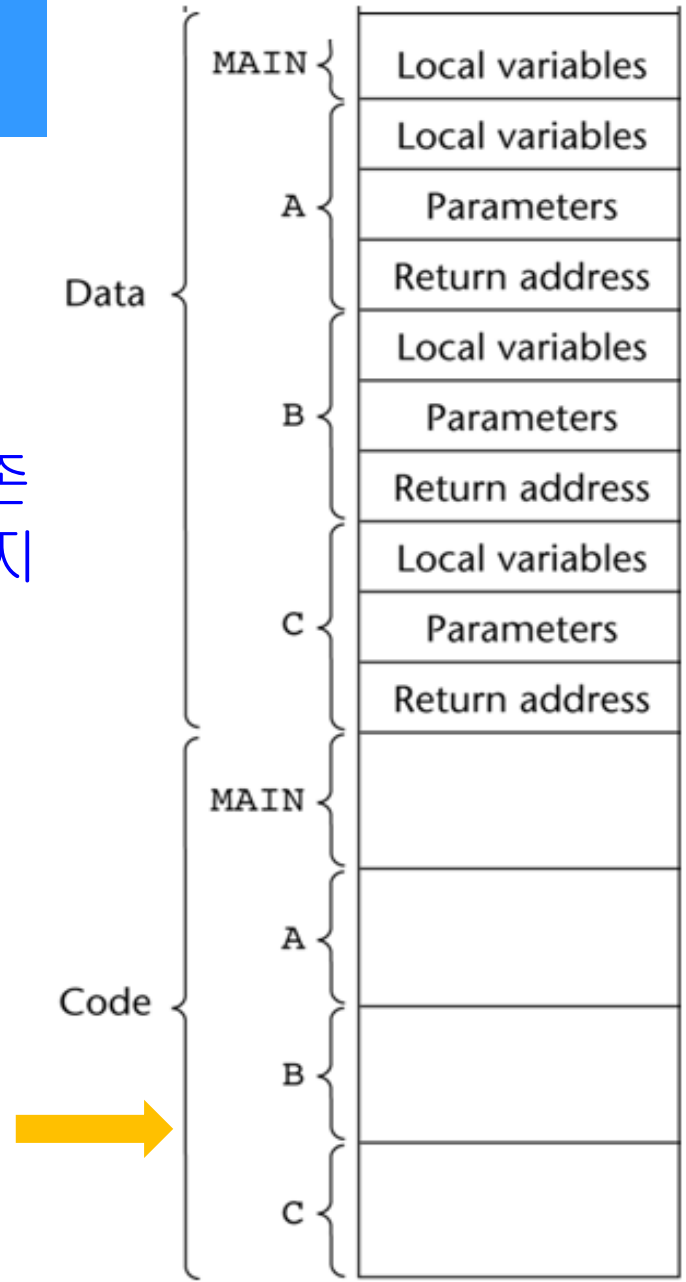
- 단순 부프로그램의 활성 레코드

Local variables
Parameters
Return address

# 예제

- 프로그램은 MAIN, A, B, C로 구성
- 각 부프로그램의 ARI는 고정된 크기  
이므로 정적 할당
- 부프로그램의 ARI는 단지 한 개만 존재 가능(재귀 부프로그램이 허용되지 않기 때문)

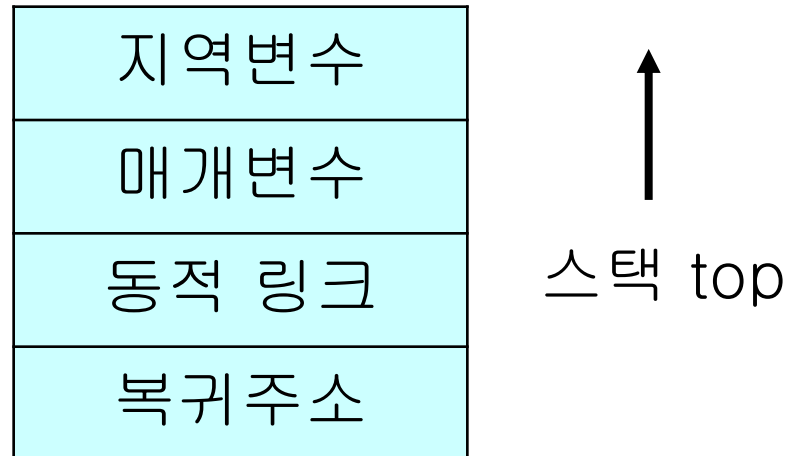
링커/로더가 MAIN이 호출될 때 구성



# 스택-동적 지역변수를 갖는 부프로그램 구현

- 스택-동적 지역변수를 갖는 부프로그램 특성
  - 스택-동적 변수 포함
  - 재귀 부프로그램 지원
- 추가 고려사항
  - 지역변수의 할당과 해제를 수행하는 코드 생성
  - 재귀 함수의 지원(부프로그램의 동시적 여러 개 활성화 지원)

# 스택-동적 변수를 갖는 언어에 대한 ARI



# 스택-동적 지역변수를 갖는 부프로그램 구현: 활성화 레코드

- 형식은 정해져 있으나(정적) 그 크기는 동적 가능 (지역 변수가 스택-동적 배열을 포함하는 경우)
- ARI는 부프로그램이 호출될 때 동적으로 생성
- ARI는 실행-시간 스택 상에 존재
- 동적 링크(dynamic link)는 호출자의 ARI의 기준에 대한 포인터
- EP(Environment Pointer)는 현재 실행 단위의 ARI의 기반 주소를 가리키며, 실행시스템에 의해서 관리 (매개변수, 지역변수의 오프셋 기준으로 사용)

# 예제

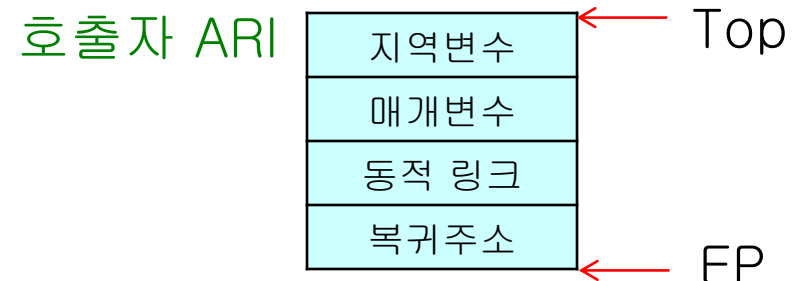
```
void sub(float total, int part)
{
    int list[5];
    float sum;
    ...
}
```

Local	sum
Local	list [4]
Local	list [3]
Local	list [2]
Local	list [1]
Local	list [0]
Parameter	part
Parameter	total
Dynamic link	
Return address	

# 부프로그램 호출/복귀 동작

- 호출자 동작

- 피호출자 ARI를 생성하여 스택에 배치
- 호출 프로그램 단위의 실행 상태 저장
- 매개변수를 평가하고 전달
- 복귀 주소를 피호출자에게 전달
- 제어를 피호출자에게 전달



# 부프로그램 호출/복귀 동작(2)

## 호출자 동작

- 피호출자 ARI를 생성하여 스택에 배치
- 호출 프로그램 단위의 실행 상태 저장
- 매개변수를 평가하고 전달
- 복귀 주소를 피호출자에게 전달
- 제어를 피호출자에게 전달

## 피호출자의 프롤로그 행동

- 현재의 EP를 동적 링크로서 스택에 저장하고, EP를 새로운 ARI의 기준 주소로 설정
- 지역 변수를 할당

피호출자 ARI



호출자 ARI

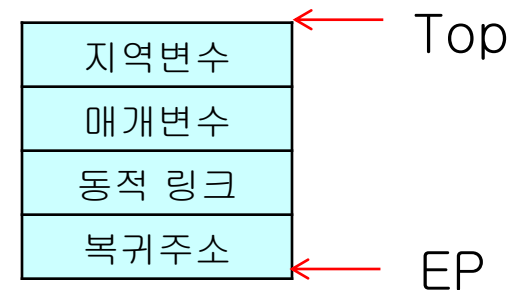
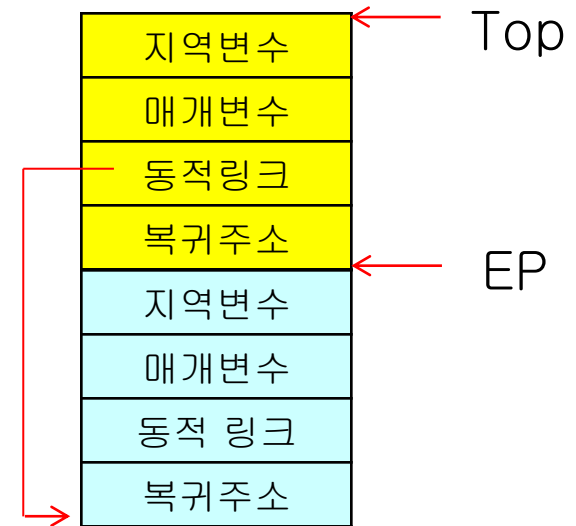




# 부프로그램 호출/복귀 동작(3)

- 피호출자의 에필로그 행동

- 값-결과-전달 매개변수나 출력-모드 매개변수 사용시, 그러한 매개변수의 값을 대응 실 매개변수로 전달
- 부프로그램이 함수이면, 함수 값은 호출자에게 접근 가능한 장소로 이동
- 스택 TOP 포인터를 현재의 EP - 1로 설정하고, EP를 피호출자 ARI의 동적 링크의 주소로 설정
- 호출자의 실행 상태를 복원
- 제어를 호출자에게 전달



## 예제: 리커전이 없는 경우

```
void fun1(float r) {  
    int s, t;  
    ... <----- 1  
    fun2(s);  
    ...  
}  
void fun2(int x) {  
    int y;  
    ... <----- 2  
    fun3(y);  
    ...  
}  
void fun3 (int q) {  
    ... <----- 3  
}  
void main () {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

각 함수의  
ARI는?

fun1의 ARI

fun2의 ARI

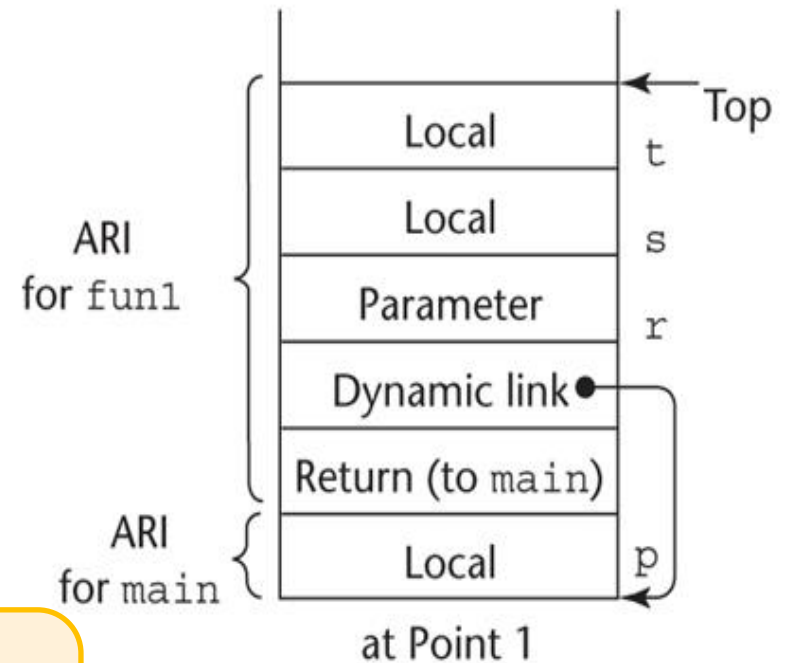
fun3의 ARI

main -> fun1 호출  
fun1 -> fun2 호출  
fun2 -> fun3 호출

## 예제: 리커전이 없는 경우 (2)

```
void fun1(float r) {  
    int s, t;  
    ...  
    fun2(s);  
    ...  
}  
void fun2(int x) {  
    int y;  
    ...  
    fun3(y);  
    ...  
}  
void fun3 (int q) {  
    ...  
}  
void main () {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

지점 1의 스택 내용



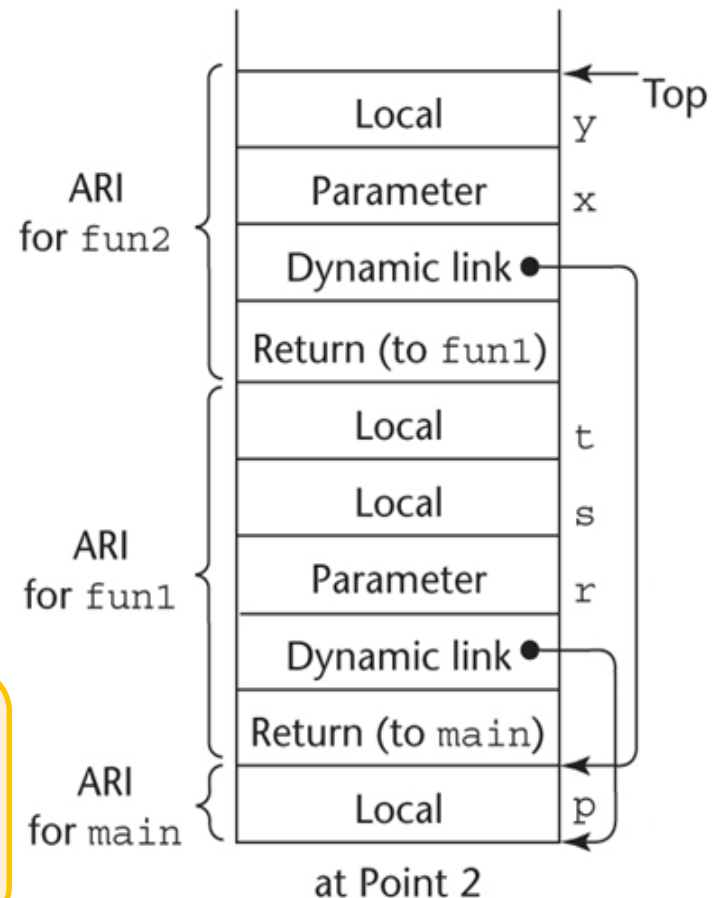
main -> fun1 호출  
fun1 -> fun2 호출  
fun2 -> fun3 호출

## 예제: 리커전이 없는 경우 (3)

```
void fun1(float r) {  
    int s, t;  
    ...                <----- 1  
    fun2(s);  
    ...  
}  
void fun2(int x) {  
    int y;  
    ...                <----- 2  
    fun3(y);  
    ...  
}  
void fun3 (int q) {  
    ...                <----- 3  
}  
void main () {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

main -> fun1 호출  
fun1 -> fun2 호출  
fun2 -> fun3 호출

### 지점 2의 스택 내용

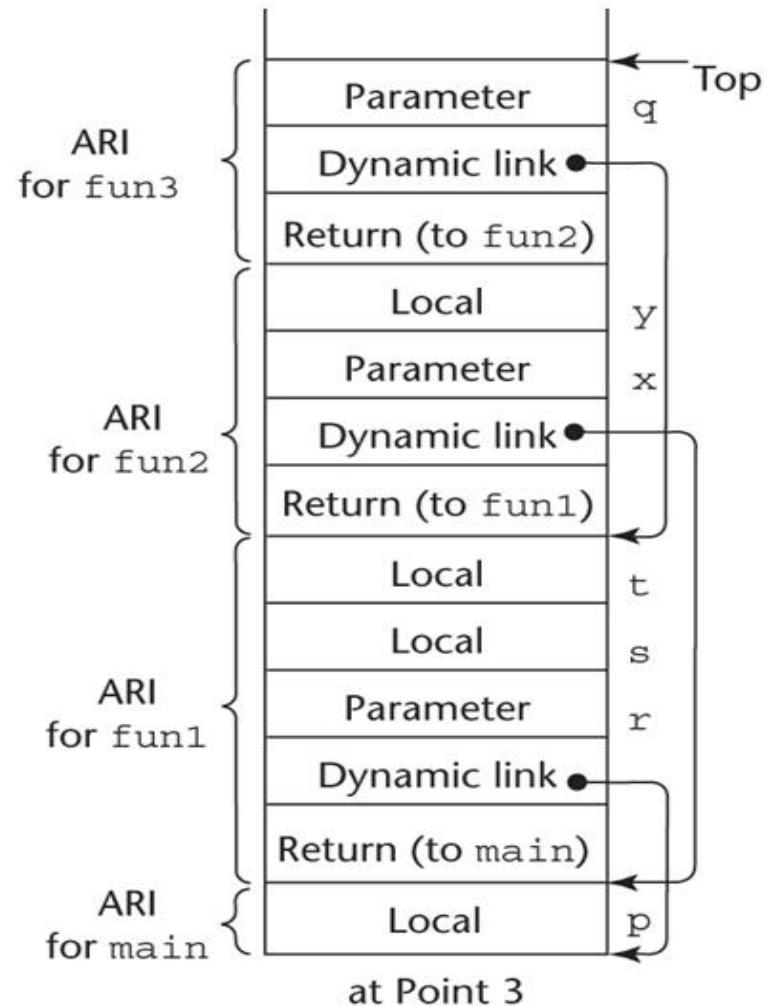


## 예제: 리커전이 없는 경우 (4)

### 지점 3의 스택 내용

```
void fun1(float r) {  
    int s, t;  
    ... <----- 1  
    fun2(s);  
    ...  
}  
void fun2(int x) {  
    int y;  
    ... <----- 2  
    fun3(y);  
    ...  
}  
void fun3 (int q) {  
    ... <----- 3  
}  
void main () {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

main -> fun1 호출  
fun1 -> fun2 호출  
fun2 -> fun3 호출

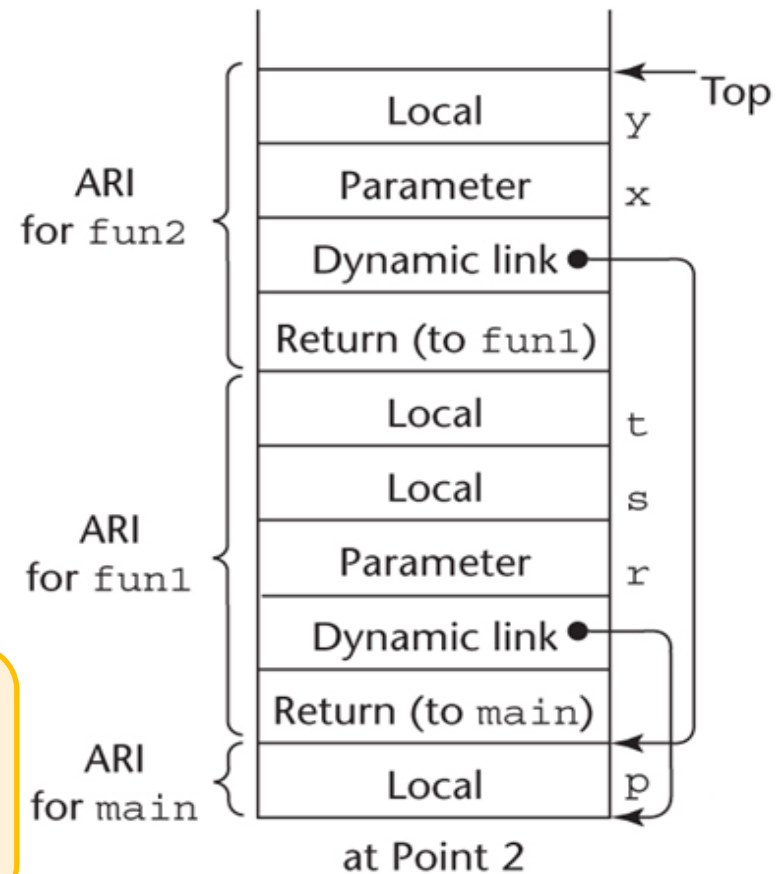


## 예제: 리커전이 없는 경우 (5)

### 지점 4의 스택 내용

```
void fun1(float r) {  
    int s, t;  
    ... <----- 1  
    fun2(s);  
    ...  
}  
void fun2(int x) {  
    int y;  
    ... <----- 2  
    fun3(y);  
    ... <----- 4  
}  
void fun3(int q) {  
    ... <----- 3  
}  
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

main -> fun1 호출  
fun1 -> fun2 호출  
fun2 -> fun3 호출

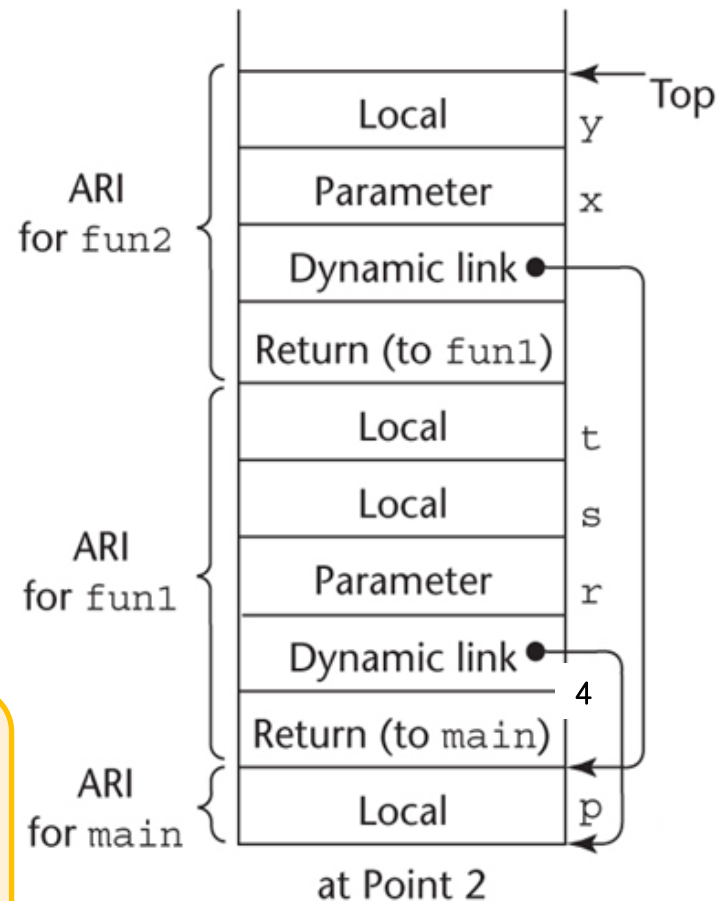


## 예제: 리커전이 없는 경우 (6)

지점 5의 스택 내용

```
void fun1(float r) {  
    int s, t;  
    ... <----- 1  
    fun2(s);  
    ... <----- 5  
}  
  
void fun2(int x) {  
    int y;  
    ... <----- 2  
    fun3(y);  
    ... <----- 4  
}  
  
void fun3(int q) {  
    ... <----- 3  
}  
  
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

main -> fun1 호출  
fun1 -> fun2 호출  
fun2 -> fun3 호출



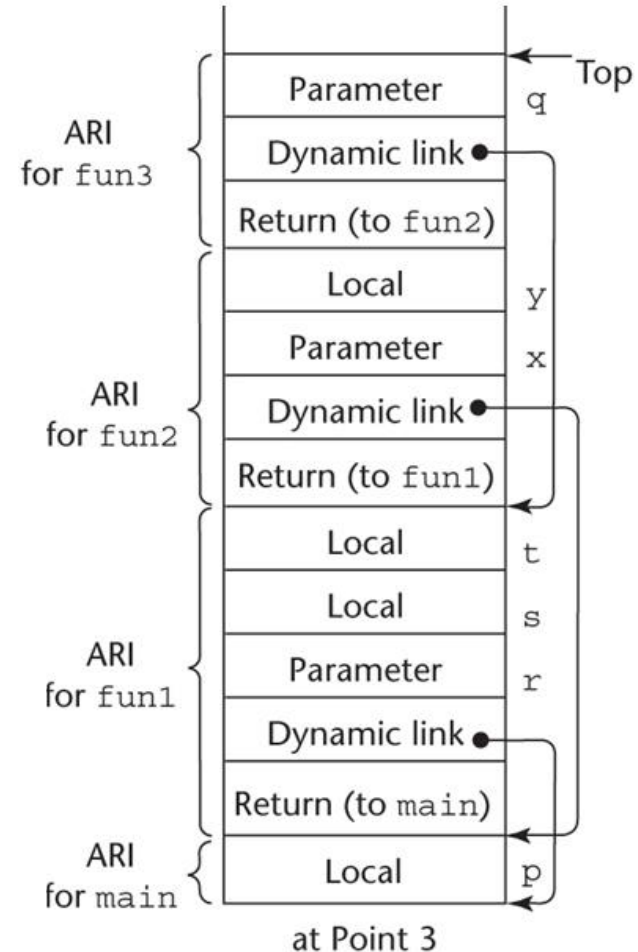
# 동적 체인과 지역-오프셋

- 동적 체인(dynamic chain)

- 주어진 시점에서 스택에 포함된 동적 링크들의 집합
- 실행의 현 시점까지 도달한 동적 history를 반영
- 호출 체인(call chain)이라고도 함

- 지역-오프셋(local-offset)

- 활성화 레코드의 시작 지점(EP)으로부터 지역변수의 오프셋을 말함
- 지역 변수의 오프셋은 컴파일러가 결정
- 지역변수의 오프셋은 ARI 시작 지점(EP)으로부터 (매개변수 개수 + 2)의 위치부터 할당





```

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

void fun2(int x) {
    int y;
    ...
    fun3(y);
    ...
}

void fun3(int q) {
    ...
}

void main() {
    float p;
    ...
    fun1(p);
    ...
}

```

fun1에서 s, t의  
지역-오프셋은?

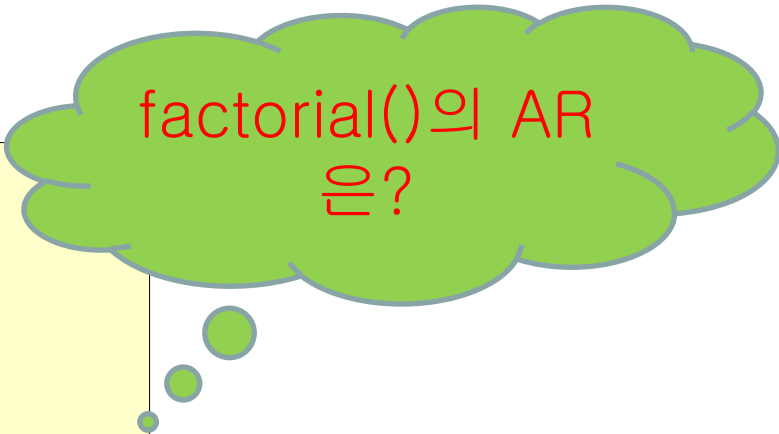


# 재귀 함수 구현

- 예제 프로그램

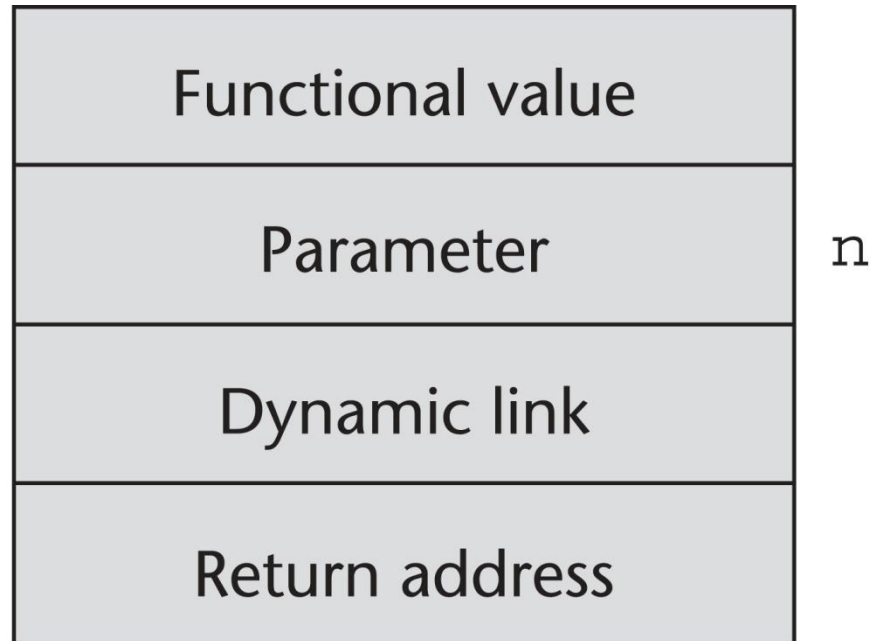
```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```



factorial()의 AR  
은?

# factorial에 대한 활성화 레코드



# factorial 호출/종료시 스택 상태

- factorial()은 3번 호출됨
- 각 호출에 대해서
  - 함수 시작 지점 1에서의 스택 상태는?
  - 함수 종료 직전 지점 2에서의 스택상태는?

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

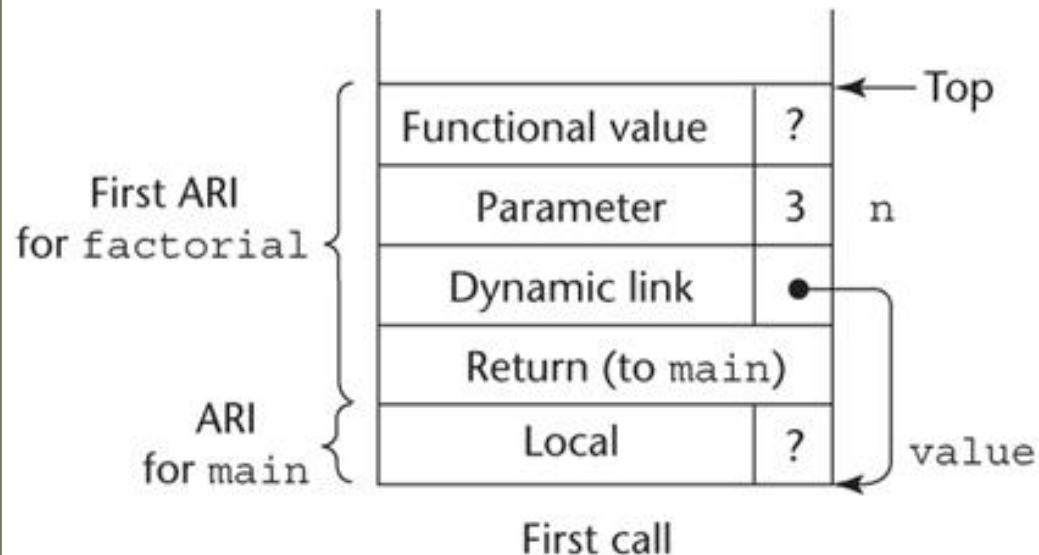
```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```

# factorial 호출시 스택 상태 (1)

- 지점 1: 첫번째 호출

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```

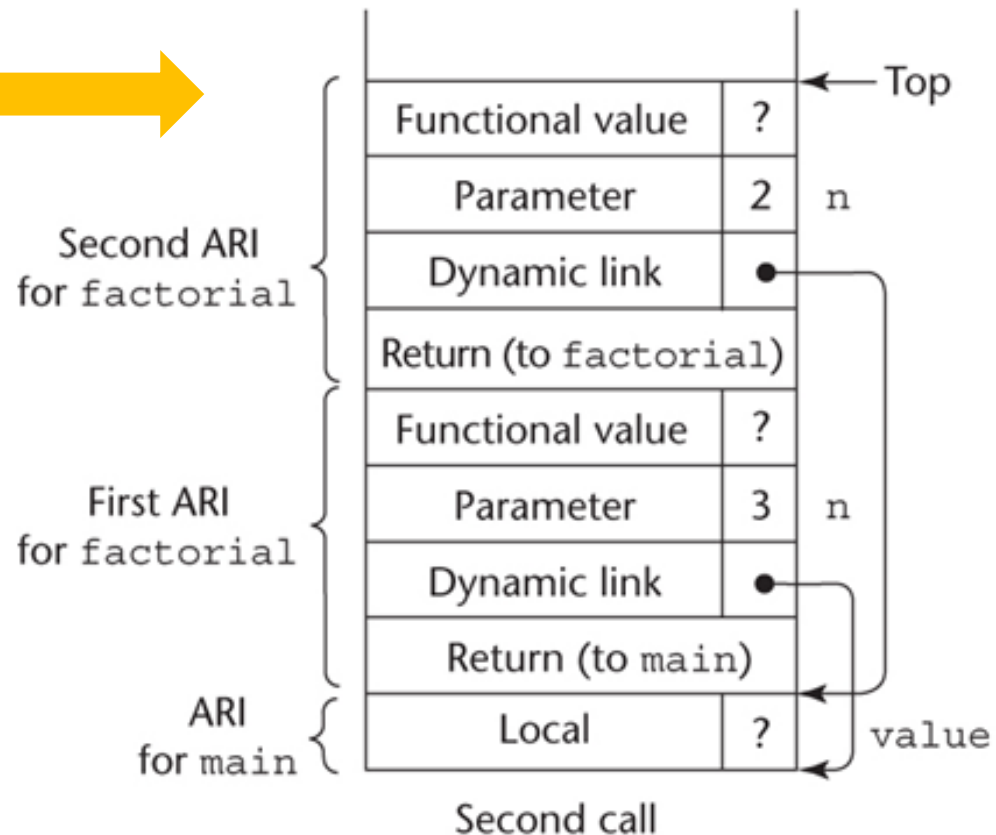


# factorial 호출시 스택 상태(2)

- 지점 1: 두번째 호출

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```

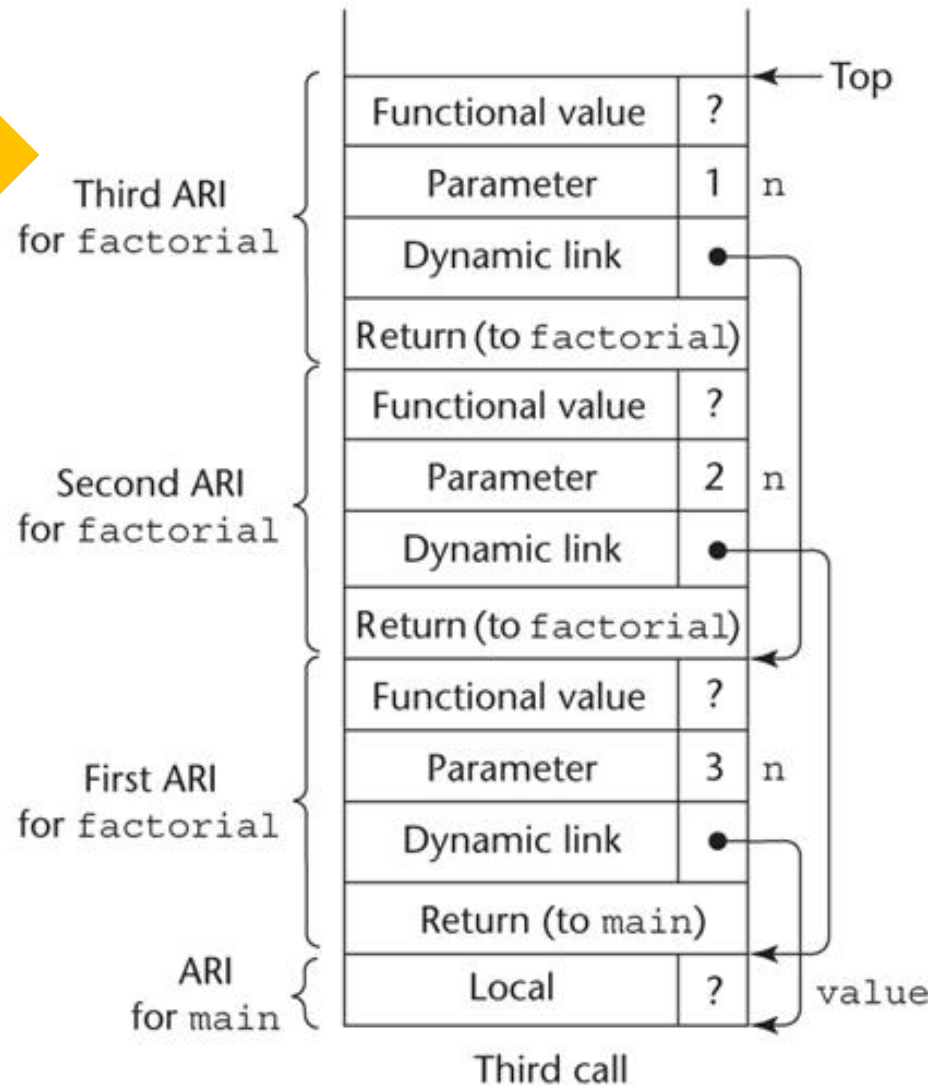


# factorial 호출시 스택 상태(3)

- 지점 1: 세번째 호출

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```

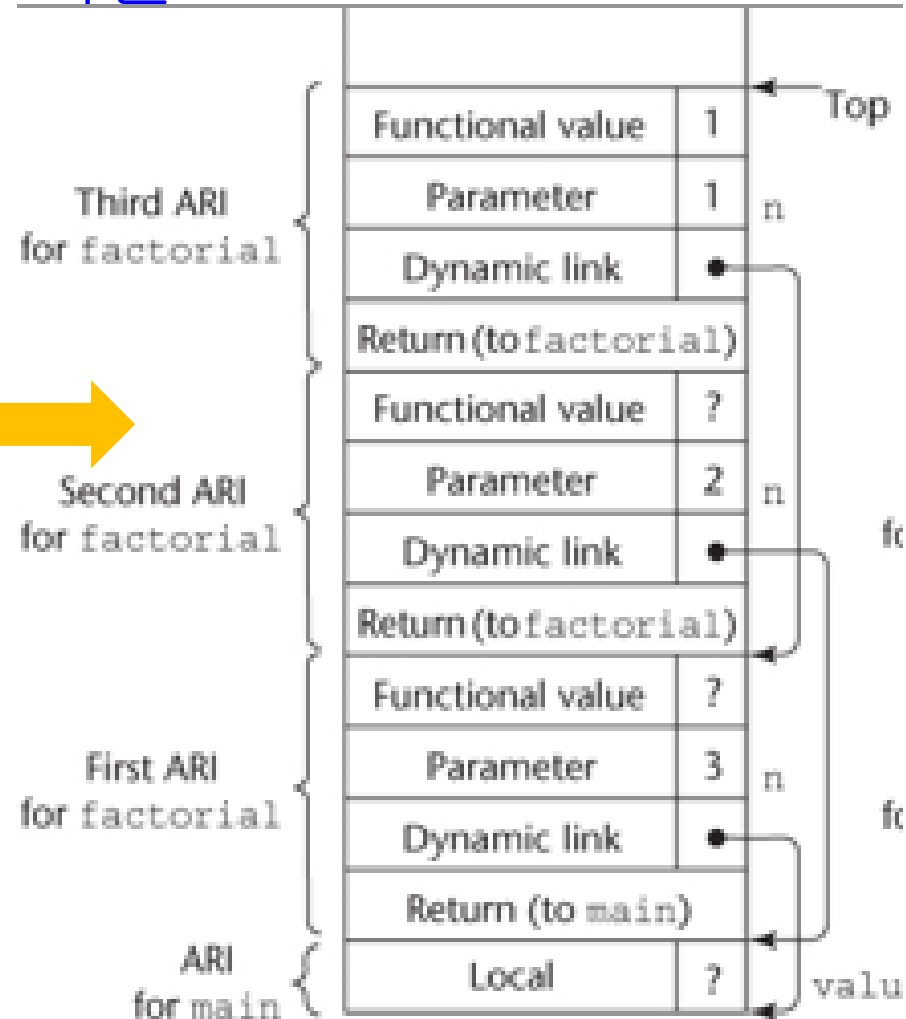


# factorial 종료 직전 스택 상태

- 지점 2: 세번째 호출 종료 직전

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```



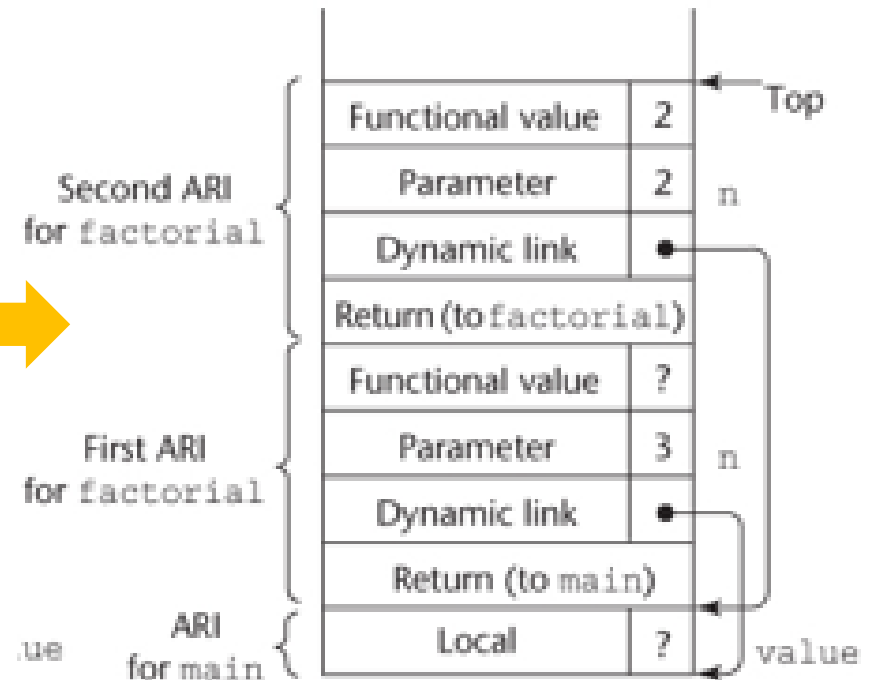


# factorial 종료 직전 스택 상태(2)

- 지점 2: 두번째 호출 종료 직전

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```

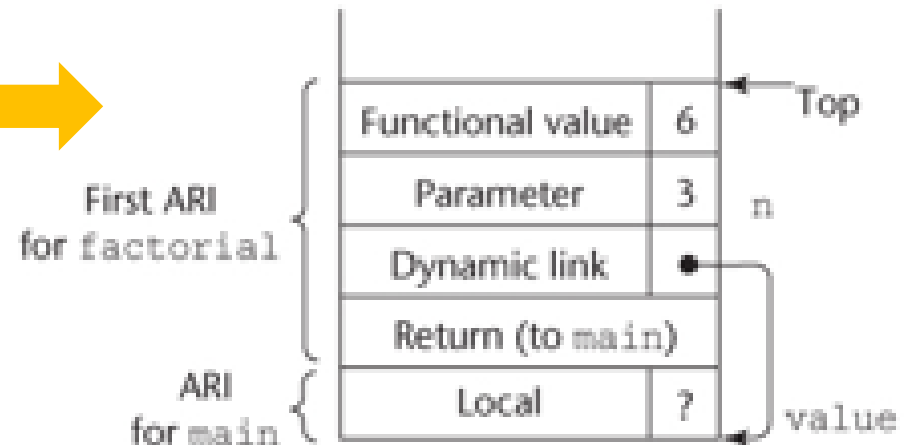


# factorial 종료 직전 스택 상태(3)

- 지점 2: 세번째 호출 종료 직전

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```

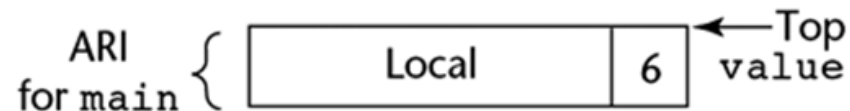


# factorial 종료 직전 스택 상태(4)

- 지점 3:

```
int factorial(int n) {  
    <----- 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n-1));  
    <----- 2  
}
```

```
void main() {  
    int value;  
    value = factorial(3);  
    <----- 3  
}
```



# 중첩 부프로그램 구현

- C 기반이 아닌 정적-영역 언어들중에서
  - 중첩 부프로그램을 허용하고,
  - 스택-동적 지역변수를 사용
  - Ex. Python, JavaScript 등
- 참조될 수 있는 모든 비지역 변수는 스택상의 ARI 상에 존재
  - 부프로그램은 모든 정적 조상 부프로그램이 활성화될 때만이 호출 가능 (정적 영역 의미 규칙)
- 비지역 변수를 찾는 과정
  - 그 변수가 할당된 올바른 ARI를 찾고,
  - 그 ARI에서 올바른 오프셋을 결정

# 정적 영역

- 정적 체인(static chain)은 스택상의 ARI들을 연결하는 정적 링크들의 체인이다.
- 정적 링크(static link)는 ARI에 포함되며, 정적 부모의 ARI의 기반을 가리킨다.
- 정적 체인은 모든 정적 선조들의 ARI를 연결한다.
- 정적-깊이(static\_depth)은 정적 영역이 최외곽 영역으로부터 얼마나 깊게 중첩되었는가를 나타내는 정수로 영역의 중첩 깊이에 해당됨
  - 중첩되지 않은 프로그램 단위의 정적-깊이는 0.

## 정적 체인(2)

- 비지역 변수의 체인-오프셋(chain-offset) 또는 중첩-깊이(nesting-depth)는 그 지역변수가 참조된 영역의 정적 깊이와 그 변수가 선언된 영역의 정적 깊이의 차이이다.
  - 비지역변수 참조를 위해 올바른 ARI에 도달하는데 필요한 체인의 길이에 해당
- 비지역 변수에 대한 참조는 (체인-오프셋, 지역-오프셋)으로 표현

# 예제

```
procedure Main_2 is
  X : Integer;
  procedure Bigsub is
    A, B, C : Integer;
    procedure Sub1 is
      A, D : Integer;
      begin -- of Sub1
        A := B + C;  <-----1
      end; -- of Sub1
    procedure Sub2(X : Integer) is
      B, E : Integer;
      procedure Sub3 is
        C, E : Integer;
        begin -- of Sub3
          Sub1;
          E := B + A;  <-----2
        end; -- of Sub3
      begin -- of Sub2
        Sub3;
        A := D + E;  <-----3
      end; -- of Sub2 }
    begin -- of Bigsub
      Sub2(7);
    end; -- of Bigsub
  begin
    Bigsub;
  end; of Main_2 }
```

Main\_2 calls Bigsub  
Bigsub calls Sub2  
Sub2 calls Sub3  
Sub3 calls Sub1

```
procedure Main_2 is
```

```
  X : Integer;
```

```
  procedure Bigsub is
```

```
    A, B, C : Integer;
```

```
    procedure Sub1 is
```

```
      A, D : Integer;
```

```
      begin -- of Sub1
```

```
      A := B + C; <-----1
```

```
    end; -- of Sub1
```

```
    procedure Sub2(X : Integer) is
```

```
      B, E : Integer;
```

```
      procedure Sub3 is
```

```
        C, E : Integer;
```

```
        begin -- of Sub3
```

```
        Sub1;
```

```
        E := B + A: <-----2
```

```
        end; -- of Sub3
```

```
      begin -- of Sub2
```

```
      Sub3;
```

```
      A := D + E; <-----3
```

```
    end; -- of Sub2 }
```

```
  begin -- of Bigsub
```

```
  Sub2(7);
```

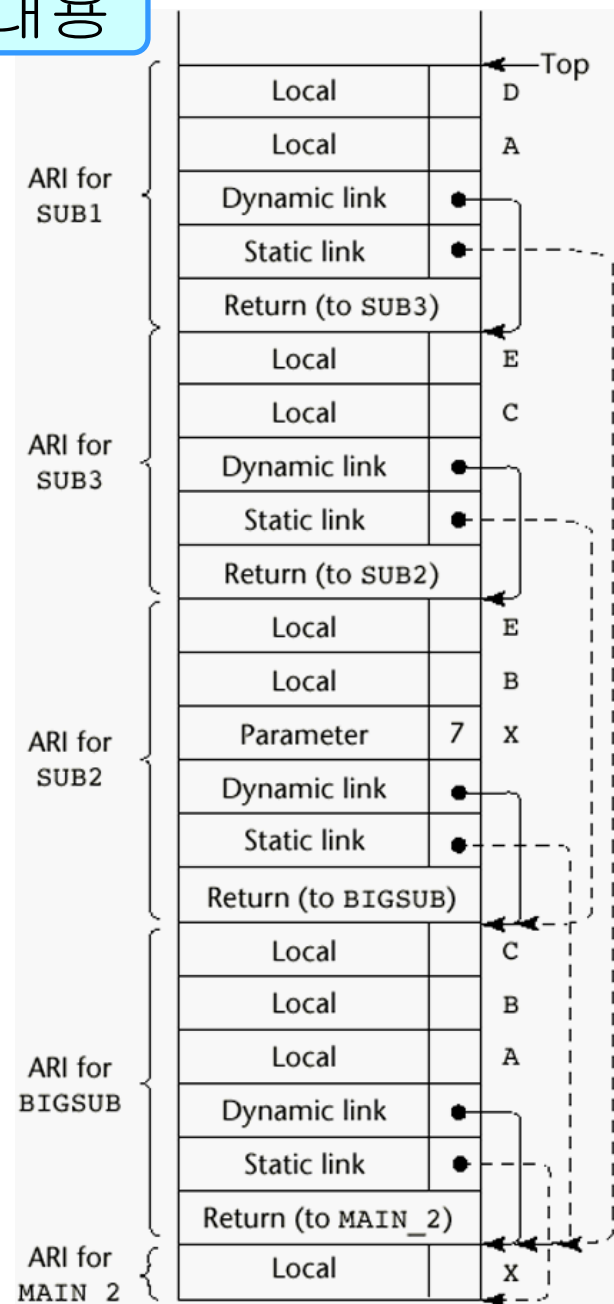
```
  end; -- of Bigsub
```

```
begin
```

```
  Bigsub;
```

```
end; of Main_2 }
```

## 지점 1의 스택 내용



Main\_2 calls Bigsub

Bigsub calls Sub2

Sub2 calls Sub3

Sub3 calls Sub1



```

procedure Main_2 is
  X : Integer;
  procedure Bigsub is
    A, B, C : Integer;
    procedure Sub1 is
      A, D : Integer;
      begin -- of Sub1
        A := B + C;  <-----1
      end; -- of Sub1
    procedure Sub2 (X : Integer) is
      B, E : Integer;
      procedure Sub3 is
        C, E : Integer;
        begin -- of Sub3
          Sub1;
          E := B + A;  <-----2
        end; -- of Sub3
      begin -- of Sub2
        Sub3;
        A := D + E;  <-----3
      end; -- of Sub2 }
    begin -- of Bigsub
      Sub2 (7);
    end; -- of Bigsub
  begin
    Bigsub;
  end; of Main_2 }

```

지점 1에서,  
A의 참조는?  
B의 참조는?  
C의 참조는?

Main\_2 calls Bigsub  
Bigsub calls Sub2  
Sub2 calls Sub3  
Sub3 calls Sub1

## 지점 1의 스택 내용

