# 프로그래밍 언어론

## Nested Subprogram의 구현

컴퓨터공학과
이만호

충남대학교
CHUNGNAM NATIONAL UNIVERSITY

# Nested Subprogram 의 구현

## 학습목표

- Nested Subprogram을 허용하는 언어에서 Subprogram을 구현하는 방법에 대해서 학습한다.

## 학습내용

- Subprogram 의 지역변수 찾기 (Static/Dynamic Scope Rule)
- Static Depth
- Static Link
- Static Chain
- Chain-Offset
- Dynamic Chain
- Block

# 목 차

- 들어가기
- 학습하기
  - Nested Subprogram
  - Static Chain
  - Dynamic Chain
  - Block
  - Dynamic Scope Rule 구현하기
- 평가하기
- 정리하기

# 알고가기 1

**Static scope rule을 적용하는 언어로 작성된오른쪽에 주어진 프로그램 골격에서, 호출 순서가 M → E → F → A → D 일 때 $$$로 표기된 부분을 실행할 때 사용 가능한 변수를 모두 나열한 것은?**

a. vm, va, vc, vd

b. vm, va, vd

c. vm, ve, vf, va, vd

d. vm, va, vf, va, vc, vd

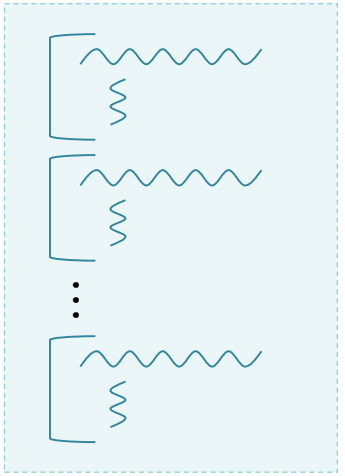
```
M
var  vm;
  A
  var  va;
    C
    var  vc;

    D
    var  vd;

      $$$

  E
  var  ve;
    F
    var  vf;
```

확인

# Nested Subprogram

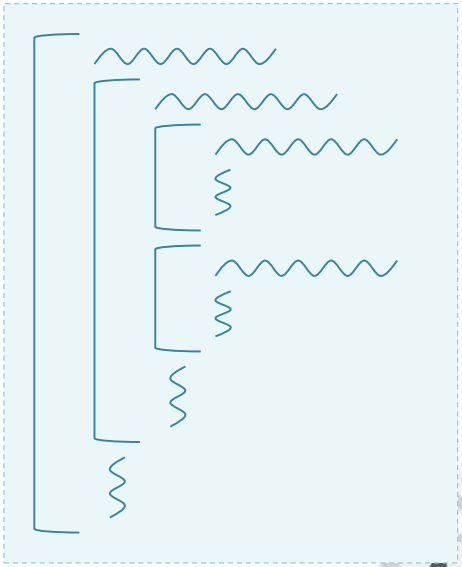
## Subprogram의 Nesting(중첩)을 허용하는 언어

- C에 기반하지 않음
- Static scope rule(정적영역 규칙)을 적용하는 언어
  - Fortran 95, Ada, Python, JavaScript, Ruby, Lua
  - 지역변수는 stack-dynamic 변수이다.



## Subprogram에서 접근 가능한 비지역변수

- 반드시 RTStack(실행시간스택)의 어떤 ARI(활성 레코드사례) 안에 존재한다.
- 비지역변수 참조(reference)하기
  - Static scope rule을 적용하는 경우를 우선 다룸
    1. 참조하고자 하는 비지역변수를 포함하고 있는 ARI를 찾아야 한다.
       - **Static chain**(정적체인)을 이용하는 방법
       - **Display**를 이용하는 방법
    2. 찾은 ARI 안에서 정확한 local-offset을 알아야 한다.
       - 비교적 쉬운 작업임
  - Dynamic scope rule을 적용하는 경우는 나중에 다룸



# Static Chain

## Static Link(정적링크)

- Static scope rule을 적용하는 언어에서, Subprogram의 ARI 안에 있어야 하는 field이다.
- Subprogram의 static parent(정적부모)의 ARI의 바닥을 point한다.

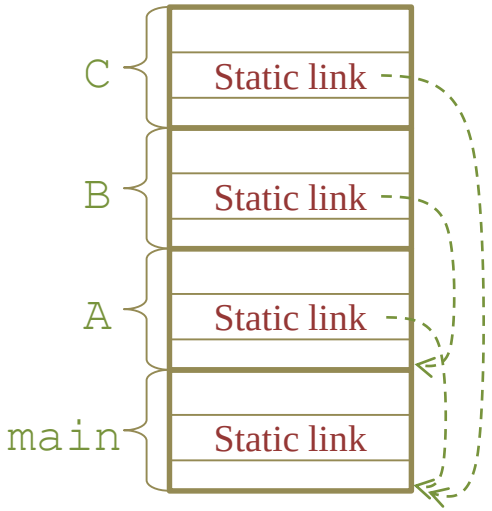
## Static Chain(정적체인)

- Static link의 연속된 연결
- 모든 static ancestor를 static parent로부터 static link를 따라 연결된 것

## Static Depth(정적깊이)

- Subprogram이 nested(중첩)되어 있는 정도를 나타낸다.
- Static ancestor의 개수와 같다.

```
main ---- static-depth: 0
  A ------ static-depth: 1
    B ----- static-depth: 2
      call C
    end B
    call B
  end A
  C ------ static-depth: 1

  end C
  call A
end main
```

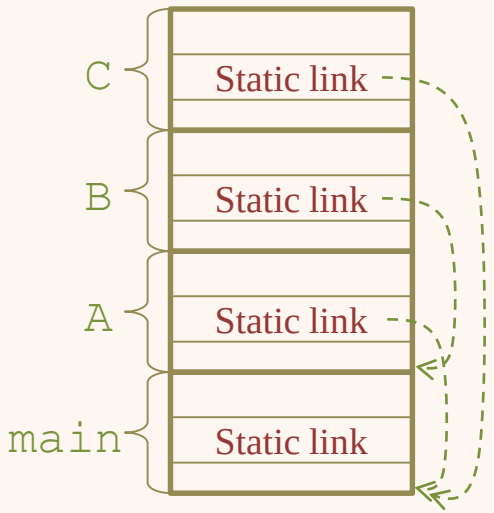


# Static Chain과 비지역변수

## Static Chain을 이용한 비지역변수 찾기

- 접근하고자 하는 비지역변수를 포함하고 있는 ARI를 찾을 때까지, 현재의 ARI로부터 static chain을 따라간다.

```
main ---- static-depth: 0
  A ------ static-depth: 1
    B ----- static-depth: 2
      call C
    end B
    call B
  end A
  C ------ static-depth: 1

  end C
  call A
end main
```



# Chain-offset과 Local-offset

## 변수의 Chain-offset

- Nesting-depth라고도 함
- "변수를 참조한 block의 static depth"와 "참조한 변수를 선언한 block의 static depth"의 차(difference)

## 변수의 Local-offset

- 변수가 포함된 ARI 안에서 변수가 EP(환경포인터)로부터 떨어져 있는 거리

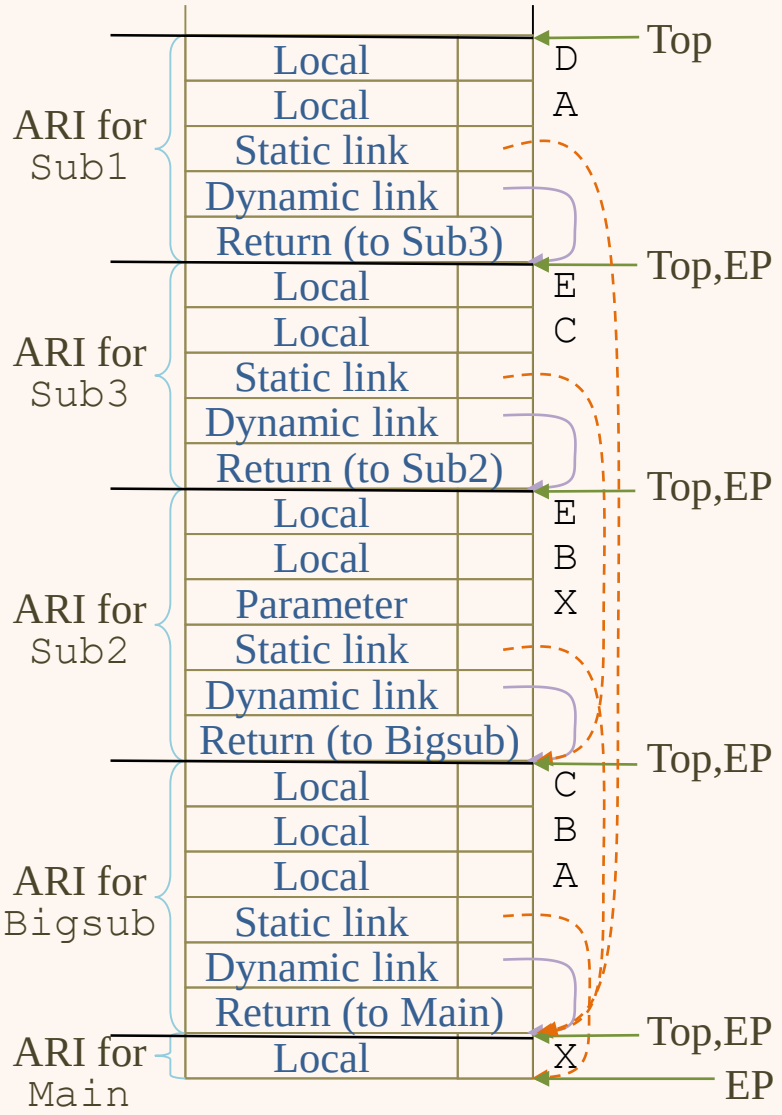
## 변수의 표현

- (chain-offset, local-offset)

```
Main {sd:0}
  var X;
  proc Bigsub(); {sd:1}
    var A, B, C;
    proc Sub1() {sd:2}
      var A, D;
      A := B + C;   {A:0,B:1,C:1}
    end;
    proc Sub2(X) {sd:2}
      var B, E;
      proc Sub3() {sd:3}
        var C, E;
        Sub1();
        E := B + A; {E:0,B:1,A:2}
      end;
      Sub3();
      A := D + E; {A:1,D:error,E:0}
    end;
    Sub2(7);
  end;
  X := 9; {X:0}
  Bigsub();
end.
```

# Chain-offset, Local-offset과 ARI

```
Main {sd:0}
  var X;
  proc Bigsub(); {sd:1}
    var A, B, C;
    proc Sub1() {sd:2}
      var A, D;
      A := B + C;   {A:0,B:1,C:1}
    end;
    proc Sub2(X) {sd:2}
      var B, E;
      proc Sub3() {sd:3}
        var C, E;
        Sub1();
        E := B + A; {E:0,B:1,A:2}
      end;
      Sub3();
      A := D + E; {A:1,D:error,E:0}
    end;
    Sub2(7);
  end;
  X := 9; {X:0}
  Bigsub();
end.
```



# Static Chain의 Maintenance

## Subprogram H(호출자)가 Subprogram P(피호출자)를 호출할 때

- P의 ARI(P-ARI라고 하자)를 구성하여 RTStack에 생성
- P-ARI의 dynamic link에 H의 ARI의 꼭대기(이전 RTStack의 Top) 주소를 저장한다.
- P-ARI의 static link에 P의 static parent의 가장 최근 ARI의 바닥 주소를 저장한다.

## 피호출자(P)의 static parent 찾기

### 방법-1

- Dynamic chain을 따라 가며 P의 첫 째 static parent를 찾는다.
- 구현하기 쉬운 방법이나, dynamic chain 탐색에 시간이 걸린다.

### 방법-2

- Subprogram의 선언과 호출을 변수의 선언과 참조처럼 생각하고 처리한다.

1. Chain-offset C = (H의 static depth) – (P를 선언한 subprogram의 static depth)
2. H-ARI의 static link로부터, C회 static chain을 따라가서 도달하는 ARI가 P의 static parent가 사용하는 ARI이다. (X라고 하자)
3. P-ARI의 static link에 X의 바닥 주소를 저장한다.

# Static Chain Maintenance - 예

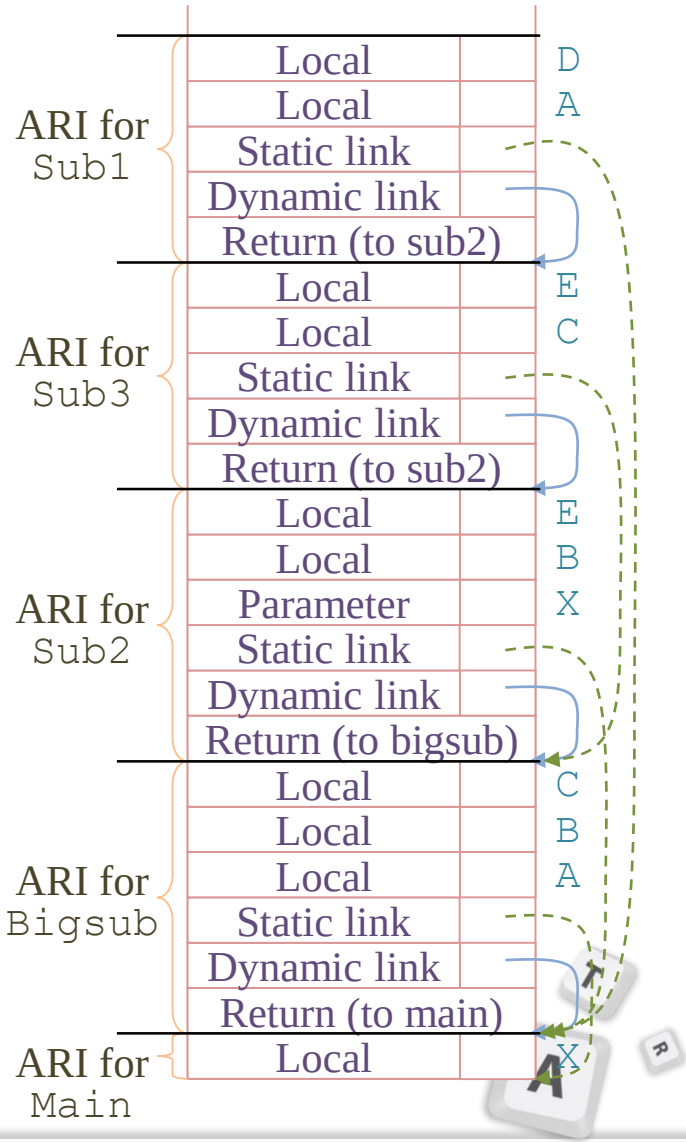
```
program Main;
  var X;
  proc Bigsub();
    var A, B, C;
    procedure Sub1();
      var A, D;
      begin
        A := B + C;
      end;
    proc Sub2(X);
      var B, E;
      proc Sub3();
        var C, E;
        begin
          Sub1();
          E := B + A;
        end;
      begin
        Sub3();
        A := D + E;
      end;
    begin
      Sub2(7);
    end;
  begin
    Bigsub();
  end.
```

**Sub2 calls Sub3**

- sd of Sub2(caller): 2
- Sub3(callee) is declared in Sub2
  sd of Sub2(parent of callee):2
- Nesting-depth: 2 – 2 = 0.
- Set static link for Sub3
  to the ARI of Sub2

**Sub3 calls Sub1**

- sd of Sub3(caller): 3
- Sub1(callee) is declared in Bigsub
  sd of Bigsub(parent of callee):1
- Nesting-depth: 3 – 1 = 2
- Follow the static chain from Sub3
  2 links.
- Set static link for Sub1
  to the ARI of Bigsub



# Static Chain Maintenance - 예

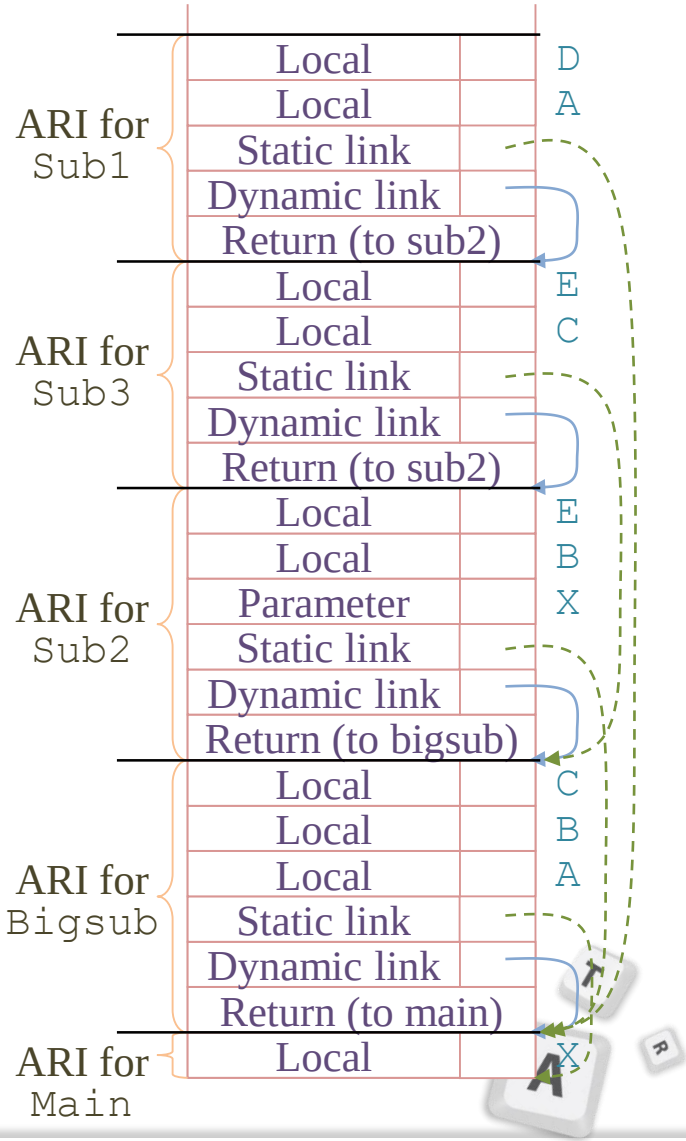
```
program Main;
  var X;
  proc Bigsub();
    var A, B, C;
    procedure Sub1();
      var A, D;
      begin
       A := B + C;
      end;
    proc Sub2(X);
      var B, E;
      proc Sub3();
        var C, E;
        begin
         Sub1();
         E := B + A;
        end;
      begin
       Sub3();
       A := D + E;
      end;
    begin
     Sub2(7);
    end;
  begin
   Bigsub();
  end.
```

**Sub2 calls Sub3**

- sd of Sub2(caller): 2
- Sub3(callee) is declared in Sub2
  sd of Sub2(parent of callee):2
- Nesting-depth: 2 – 2 = 0.
- Set static link for Sub3
  to the ARI of Sub2

**Sub3 calls Sub1**

- sd of Sub3(caller): 3
- Sub1(callee) is declared in Bigsub
  sd of Bigsub(parent of callee):1
- Nesting-depth: 3 – 1 = 2
- Follow the static chain from Sub3
  2 links.
- Set static link for Sub1
  to the ARI of Bigsub



# Static Chain의 평가

## 비지역변수 참조와 선언 사이의 관계

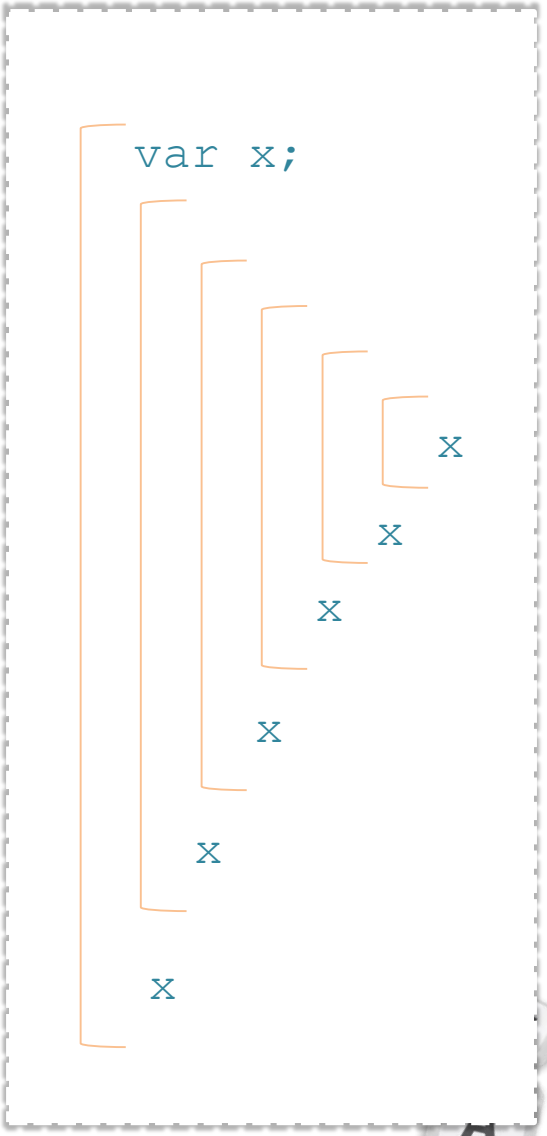
- 비지역변수를 참조(reference)하는 subprogram과 그 비지역변수가 선언(declaration)된 subprogram 사이에 중첩된 깊이가 깊을수록 참조 시간이 더 길다.

## Time-critical(시간-임계) code를 작성하기 어렵다.

- 비지역변수 참조 시간이 일정하지 않다.
- Code 수정으로 인해 nesting depth(중첩깊이)가 변할 수 있고, 따라서 참조시간도 변하게 된다.

## Static chain의 단점을 보완할 수 있는 대안

- Display 사용



# Display

## 개념

- Static link를 display라고 하는 별도의 장소에 저장
- Display는 배열로서 현재 시점의 참조환경에 있는 ARI들의 pointer가 저장된다.

## 비지역변수의 참조

- (display-offset, local-offset)으로 표시된다
  - Display-offset: static_depth와 동일
  - Local-offset: ARI 내에서의 offset이므로 static chain의 경우와 같음

## 장점

- 비지역변수 참조 시간이 일정하다.

# Display 유지관리

- Display-offset은 static depth에 의해서만 결정된다.
- 실행 중인 subprogram의 static depth가 n인 경우, display에는 (n+1)개의 값이 유효하다.
- Subprogram H가 P를 호출할 때, display 유지관리 방법

(H의 static depth: n, P의 static depth: k) 라고 할 때,

1. P의 ARI(P-ARI)를 생성
2. P-ARI 안에 display[k]를 저장
3. Display[k]가 P-ARI를 point하도록 함
4. Subprogram P 실행
5. P-ARI에 저장해둔 이전의 display[k] 값을 display[k]에 저장

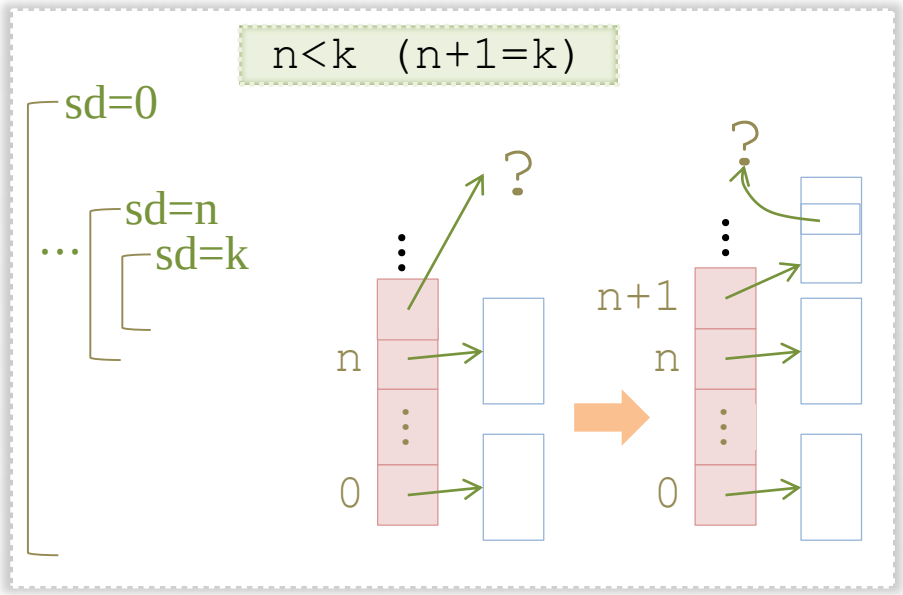
3가지 경우가 있음

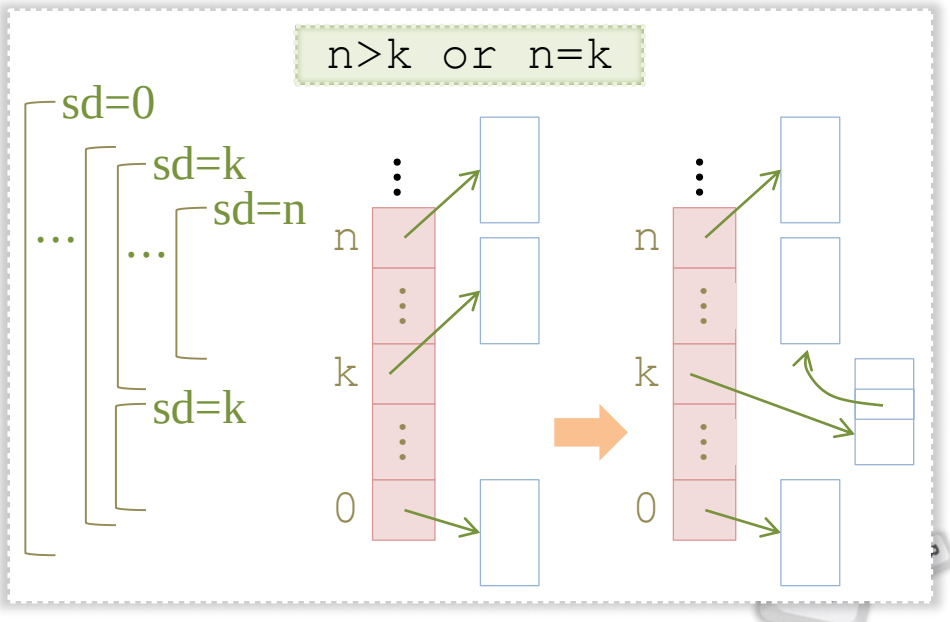
- 1. n<k (k=n+1)  2. n>k  3. n=k
- 모든 경우의 처리 과정이 동일함

# Display 유지관리 - 예

- Subprogram H가 P를 호출할 때, display 유지관리 방법
  - (H의 static depth: n, P의 static depth: k) 라고 할 때,

1. P의 ARI(P-ARI)를 생성
2. P-ARI 안에 display[k]를 저장
3. Display[k]가 P-ARI를 point하도록 함
4. Subprogram P 실행
5. P-ARI에 저장해둔 이전의 display[k] 값을 display[k]에 저장



# Static Chain vs. Display

## 지역변수 참조 : 차이 없음

## 비지역변수 참조 시간

- Display 방법에서는 일정함 → Time-critical(시간-임계) code에 좋음
- If chain-offset = 1, 별 차이 없음
- If chain-offset > 1, display가 빠름

## Subprogram을 호출하는데 걸리는 시간

- Display 방법에서는 일정함
- If chain-offset ≤ 2, static chain이 빠름
- If chain-offset > 2, display가 빠름

## Subprogram 실행을 끝내고 return하는데 걸리는 시간

- 둘 다 일정한 시간이 걸리나, static chain이 약간 빠름

## 전체적인 면에서의 평가

- Display가 register에 저장된다면, display가 효율적이다. (비현실적)
- Static chain이 더 효율적이고, 널리 쓰인다.

# Blocks

**프로그램 code 중간에 프로그래머가 지정하는 영역**

**Block 안에서 변수를 선언할 수 있다.**

- 선언된 변수는 해당 block 내에서만 사용된다.
- 선언된 변수의 lifetime: block 시작 ~ 종료

예

- C, C++, C#, Java : . . . **{ . . . . . }** . . .
- Ada : . . . **declare . . . begin . . . end** . . .

**장점**

- **Block 외부에서 선언된 같은 이름의 변수와 무관하다.**

예

C 언어

```
{ int tmp;
   tmp = list[upper];
   list[upper] = list[lower];
   list[lower] = tmp
}
```

# Block 구현하기

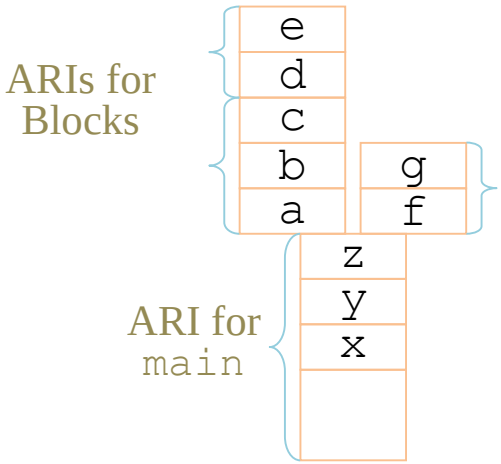
## 방법-1

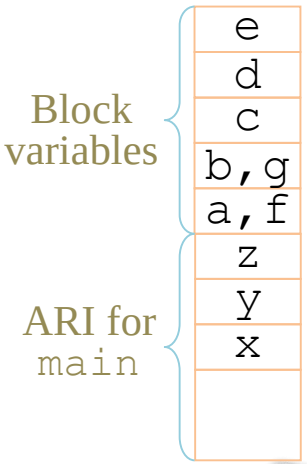
- Block을 parameter가 없는 subprogram처럼 다룬다.
  - Activation record(활성레코드)를 사용한다.

## 방법-2

- Subprogram이 현재 사용하는 ARI의 꼭대기에 block 지역변수를 할당한다.
  - Block 지역변수가 사용할 기억장소의 최대 크기를 결정한다.
  - ARI의 꼭대기에 block 지역변수가 사용할 기억장소를 할당한다.

```
void main() {
  int  x, y, z;
  while (...) {
    int  a, b, c;  ...
    while (...) {
      int  d, e;  ...
    }
  }
  while (...) {
     int  f, g;  ...
  }
  ...
}
```



# Dynamic Scope Rule 구현하기

**Dynamic scope rule을 적용하는 언어에서 비지역변수 찾기**

- Static ancestor와 관계 없음
- Subprogram을 호출한 순서의 역순으로 비지역변수를 찾음
  - 오른쪽 code에서, subprogram 호출 순서가 `M → E → F → A → D` 일 경우, `$$$` 지점을 실행할 때, 비지역변수를 찾는 순서는 `D,A,F,E,M` 임

**Dynamic scope rule을 적용하는 언어에서 비지역변수 참조 해결 방법**

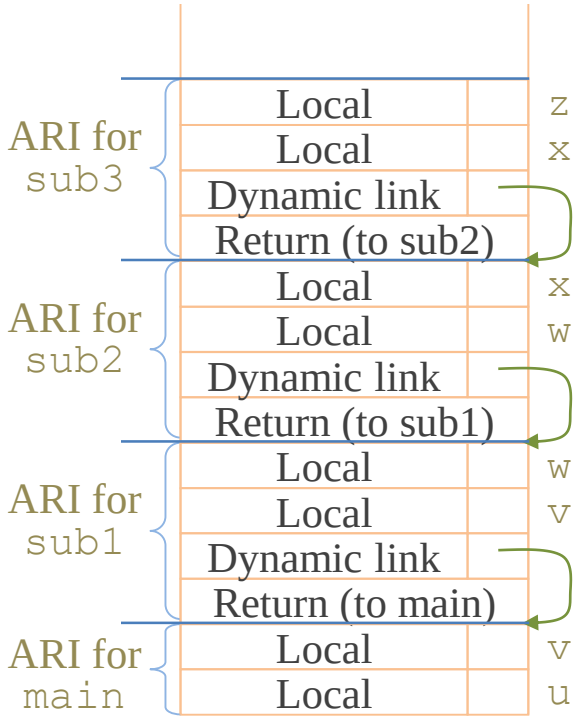
- Deep access(심층접근)
- Shallow access(피상접근)

```
M
var  vm;
  A
  var  va;
    C
    var  vc;


    D
    var  vd;

      $$$
  E
  var  ve;
    F
    var  vf;
```

# Deep Access

- 비지역변수를 dynamic chain을 따라 ARI를 탐색하여 찾는다.
- ARI는 변수이름을 가지고 있어야 한다. (static chain에서는 값만 가짐)
- Chain의 길이는 실행하기 전에는 알 수가 없다. (not static)

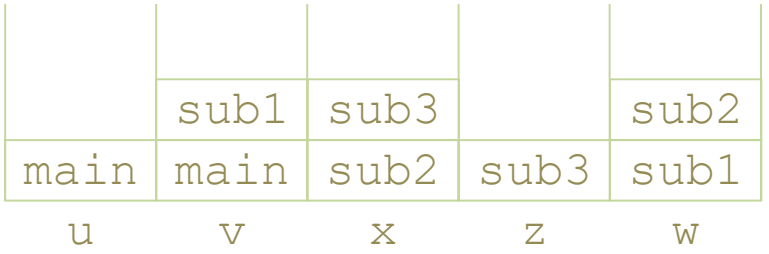
```
void sub3() {
  int x, z;  x = u + v;  ...
}

void sub2() {
  int w, x; ... sub3(); ...


void sub1() {
  int v, w; ... sub2() ...;
}

void main() {
  int u, v; ... sub1(); ...
}
```



# Shallow Access

- 각 변수 이름마다 하나의 stack을 사용
- 각 변수를 선언한 subprogram 이름을 해당 변수의 stack에 호출된 순서대로 저장

```
void sub3() {
  int x, z;  x = u + v;  ...
}

void sub2() {
  int w, x; ... sub3(); ...
}

void sub1() {
  int v, w; ... sub2(); ...
}

void main() {
  int u, v; ... sub1(); ...
}
```




변수에 대한 참조는 해당 변수의 stack에서 꼭대기에 있는 subprogram에서 선언된 것이 참조된다.

# 평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?

총 3문제가 있습니다.

START

# 평가하기 1

**1. Subprogram(P라고 하자)이 사용하는 ARI(활성레코드 사례)의 static link에 저장되는 값은?**

a. P의 최상위 static ancestor의 ARI의 꼭대기 주소

b. P의 static parent의 ARI의 바닥 주소

c. P를 호출한 subprogram의 ARI의 바닥 주소

d. P의 static parent의 ARI의 꼭대기 주소

확인

# 평가하기 2

**2. Subprogram(P라고 하자)이 사용하는 ARI의 dynamic link에 저장되는 값은?**

a. P를 호출한 subprogram의 ARI의 꼭대기 주소

b. P를 호출한 subprogram의 ARI의 바닥 주소

c. P의 static parent의 ARI의 바닥 주소

d. P의 static parent의 ARI의 바닥 주소

확인

# 평가하기 3

**3. Static scope rule을 적용하는 언어에서, 비지역변수 참조를 구현하는 방법으로 static chain을 이용하는 경우, 비지역변수를 접근하기 위해 필요한 정보는?**

a. Static depth, Local-offset

b. Chain-offset, Local-offset

c. Chain-offset, Static depth

d. Static depth, Local-offset

확인

# 정리하기

## Nested Subprogram

## Static Chain

- Static scope rule을 적용하는 언어에서 비지역변수를 접근할 때 사용
- 필요한 개념들
  - Static Link
  - Static, Depth
  - Chain-offset

## Dynamic Chain

- Dynamic scope rule을 적용하는 언어에서 비지역변수를 접근할 때 사용

## Block

- 지역변수를 선언할 수 있다.
- Parameter가 없는 subprogram처럼 취급할 수 있다.

## Dynamic Scope Rule 구현하기

- Deep Access
- Shallow Access

"강의를 마치겠습니다. 수고하셨습니다."