Implementing Subprograms

국민대학교 컴퓨터공학부
강 승 식

Topics

- The General Semantics of Calls and Returns
- Implementing "Simple" Subprograms
- Implementing Subprograms with Stack-Dynamic Local Variables
- Nested Subprograms and Blocks
- Implementing Static Scoping

General Semantics of Calls and Returns

함수 콜할때 : 함수 F라는 놈이 함수 G를 콜했다?? 치면
실행환경을 보존해주어야함. 인자전달 메모리할당 + 로칼변수 메모리할당
그후 함수 실행 하면댐

- Calls to a subprogram:
  - Create an activation record instance
  - Save the execution status of calling program
    - Save the old EP in the stack as the dynamic link and create the new value
  - Parameter passing
    - Stack-dynamic allocation of local variables
  - Transfer of control and arrange for the return
    - Pass the return address to the called 함수 콜한거 돌아왔을 때 다음꺼 실행하기 위해
  - If subprogram nesting is supported, access to nonlocal variables must be arranged

돌아올때?? 레퍼런스일땐 필요가없는데 call by result인경우
return value 받아야하고 등등등 하고 f를 다시 실행 할 수 있는 환경으로 되돌려야함

- Subprogram returns:
  - Out mode and inout mode parameters must have their values returned
  - Deallocation of stack-dynamic locals
  - Restore the execution status
  - Return control to the caller

Implementing "Simple" Subprograms

심플에서는 recursive 불가능
모든지역변수는 static

fortran에선
시작할때 메모리 할당 마침

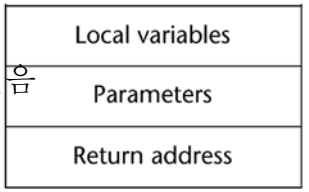
메모리가 이미 준비되어있는상태에서 실행함
이렇게 코드구현하면 쉬움 실행전에 메모리가 모두 할당됨
그러나 함수가 50개쯤있다. 근데 그중에 다 쓰는건아니다
메모리낭비임 call 할때만 메모리 할당아는게 아니기 때문

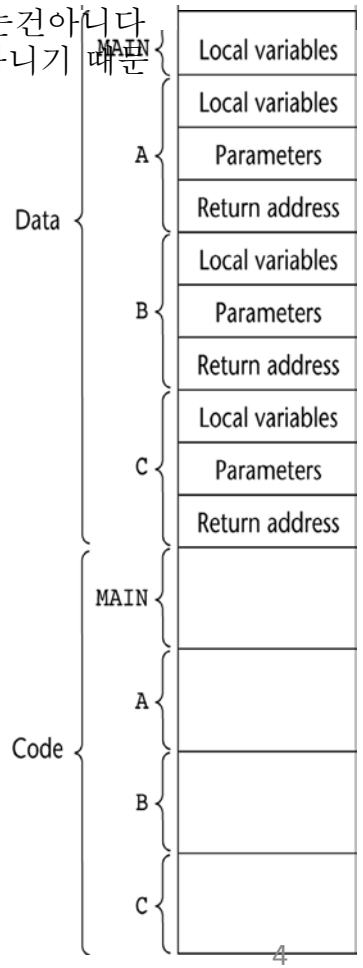
- Required storage:
  - Status information, parameters, return address, return value for functions, temporaries
- Two separate parts:
  - the actual code and the non-code part(local variables)

지역변수

- *activation record*
  - format, or layout, of the non-code part
  - An *activation record instance* is a concrete example of an activation record

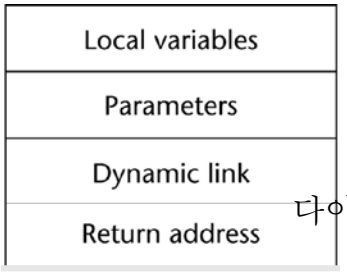
재귀적인 경우 activation instance는 여러개일수있음





Typical Activation Record for a Language with **Stack-Dynamic** Local Variables

- The *dynamic link* points to the top of an instance of the activation record of the caller
- An activation record instance is dynamically created when a subprogram is called
- Activation record instances reside on the run-time stack
- The *Environment Pointer* (EP) must be maintained by the run-time system. It always points at the base of the activation record instance of the currently executing program unit



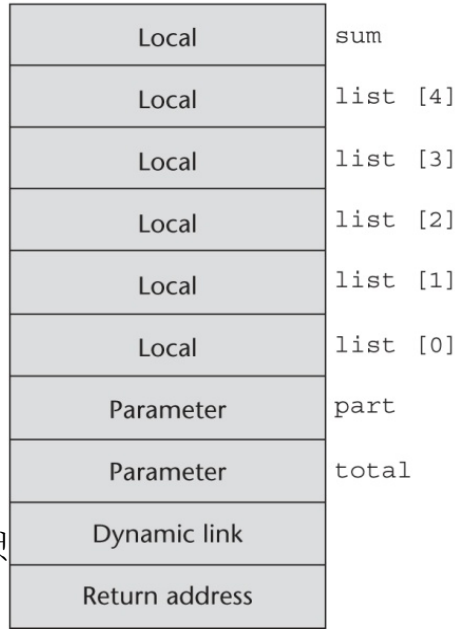
다이나믹 링크 추가댐

Dynamic link
내가 필요할 때만
메모리 할당함 끝나면 메모리 pop

base address가 정해져있지 않음 call할때마다 바뀜. 기준이필요

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

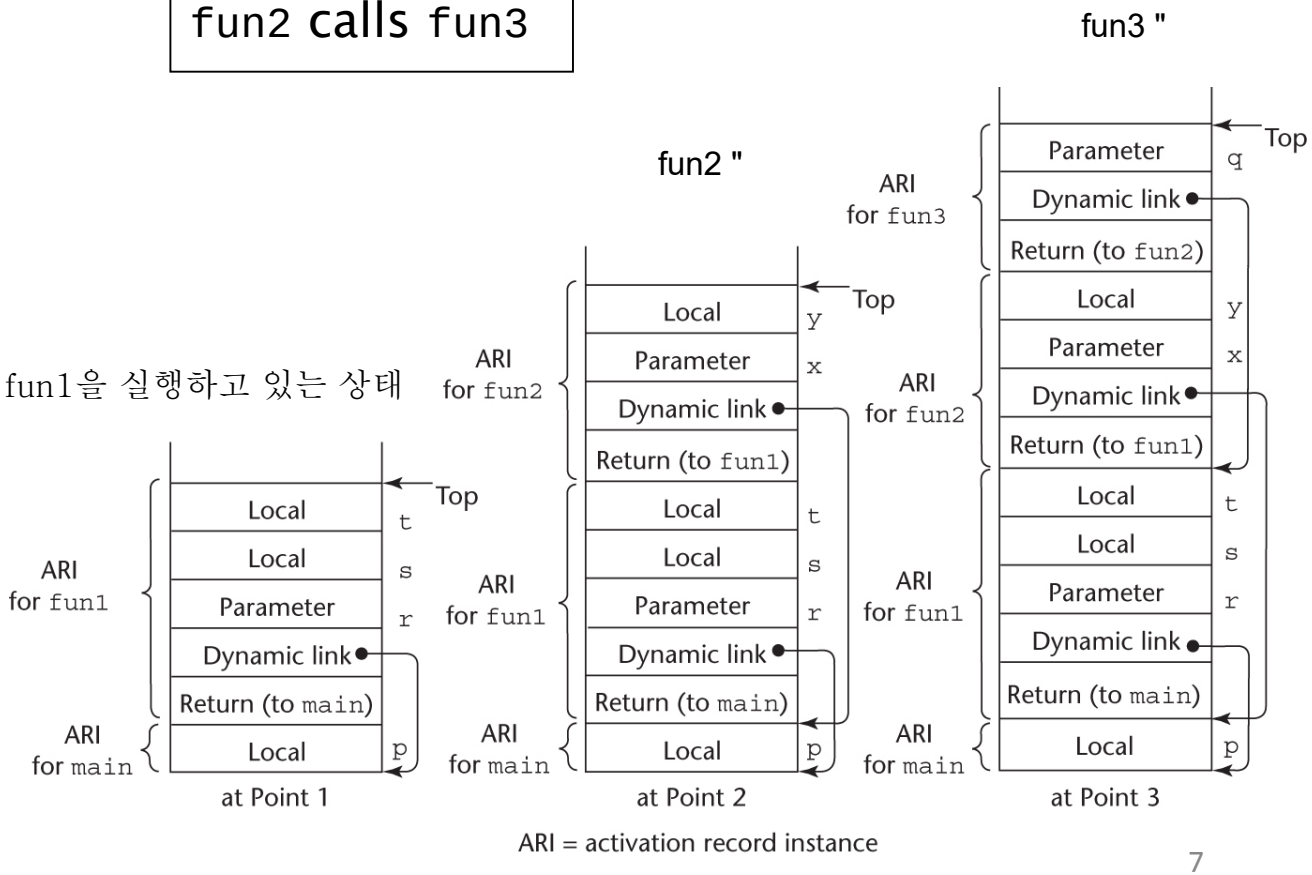
재귀적 하기위해 dynamic link 추가한것



An Example Without Recursion

```
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);
   ...
}
```

main calls fun1
fun1 calls fun2
fun2 calls fun3



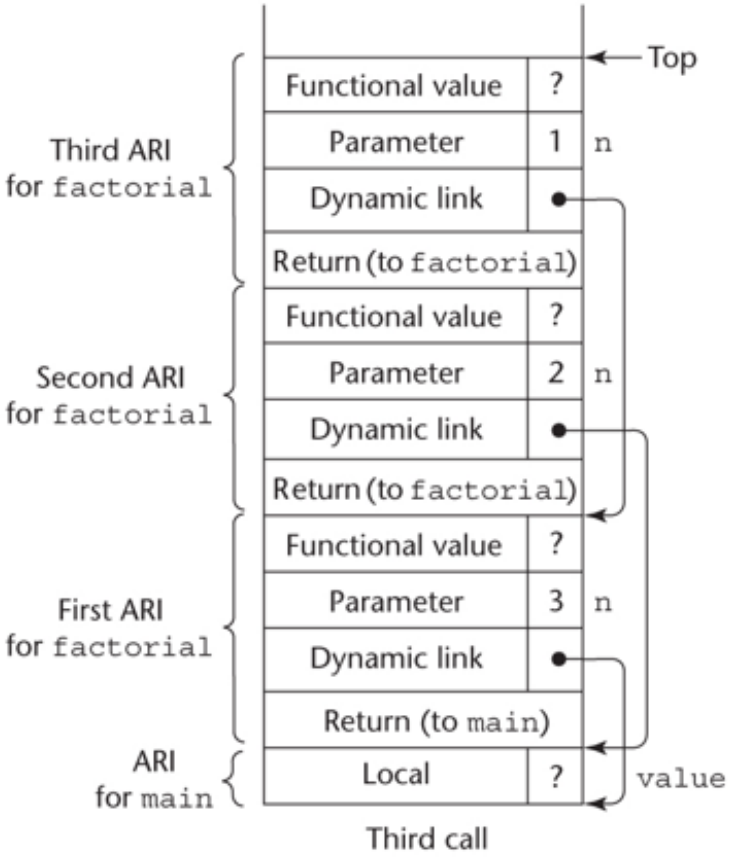
Dynamic Chain and Local Offset

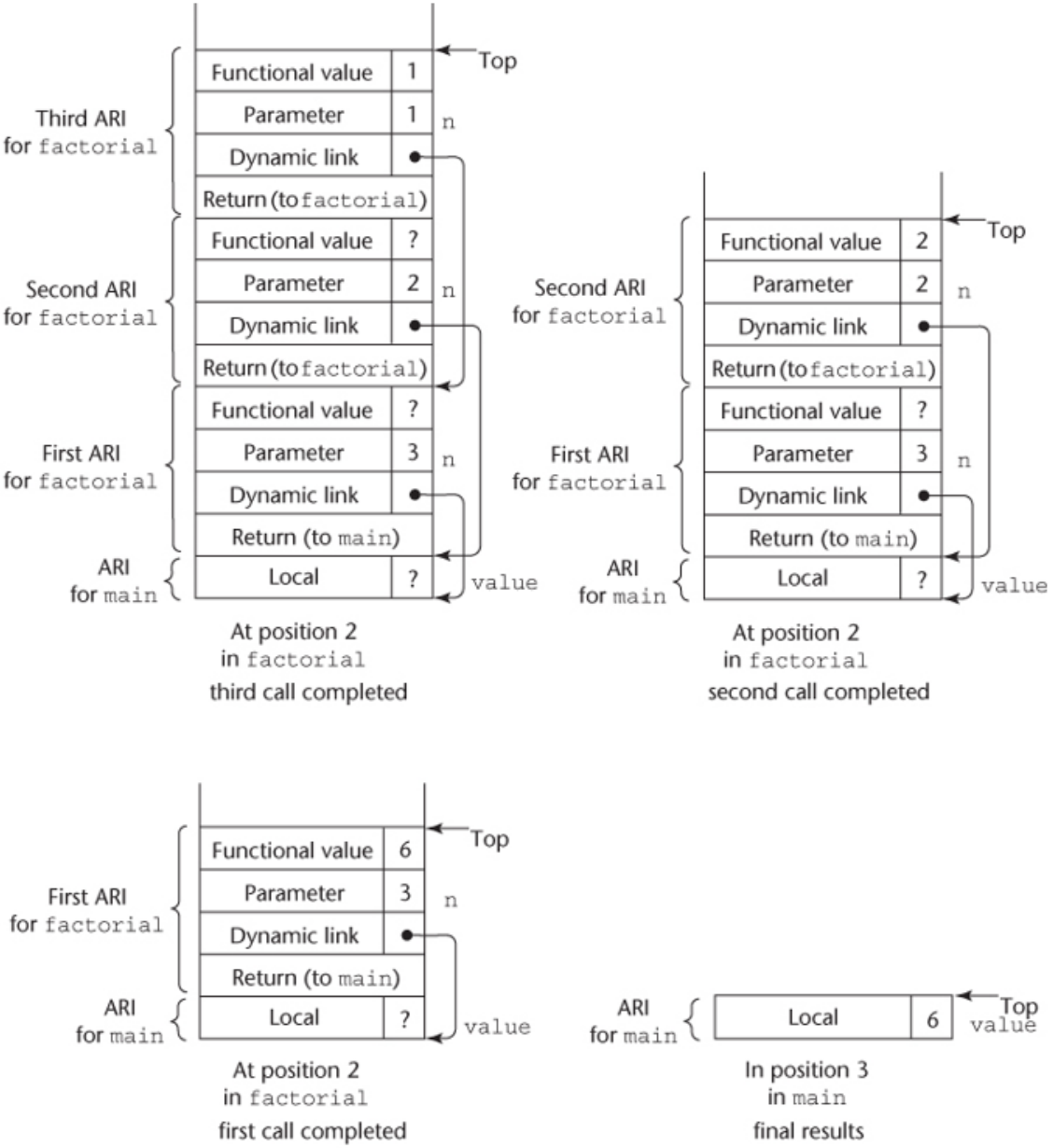
- *dynamic chain*, or *call chain*
  - Collection of dynamic links in the stack at a given time

- *local_offset*
  - Local variables can be accessed by their offset from the beginning of the activation record, whose address is in the EP.

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

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





ARI = activation record instance

Non-local Reference, Static Chain

- Finding the correct activation record instance for non-local variable
- The *static link* in an activation record instance for subprogram A points to one of the ARI of A's static parent
- The static chain from ARI connects it to all of its static ancestors

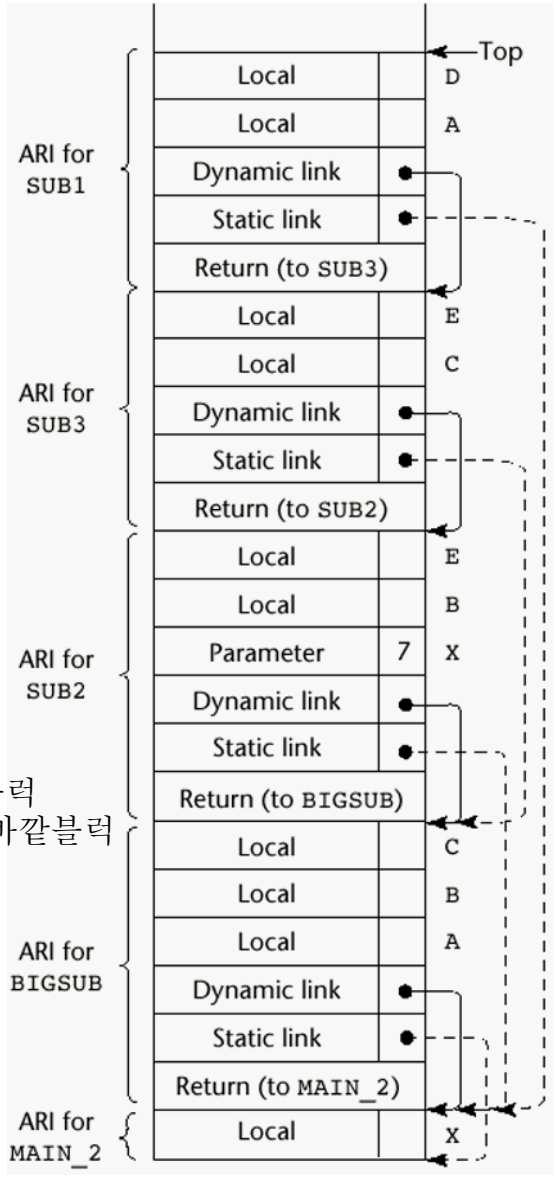
Example Ada Program

```
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 := X + E;  <-----------------------3
      end; -- of Sub2 }
    begin -- of Bigsub
    Sub2(7);
    end; -- of Bigsub
  begin
  Bigsub;
end; of Main_2 }
```

- Call sequence for `Main_2`

  `Main_2` calls `Bigsub`
  `Bigsub` calls `Sub2`
  `Sub2` calls `Sub3`
  `Sub3` calls `Sub1`

B는 바깥블럭
A는 바깥,바깥블럭
찾아감



Blocks

- Blocks are user-specified local scopes for variables
- An example in C

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

- Lifetime of `temp` begins when control enters the block

Summary

- Subprogram linkage semantics requires many action by the implementation
- Simple subprograms have relatively basic actions
- Stack-dynamic languages are more complex
- Subprograms with stack-dynamic local variables and nested subprograms have two components
  - actual code
  - activation record
- Activation record instances contain formal parameters and local variables among other things
- Static chains are the primary method of implementing accesses to non-local variables in static-scoped languages with nested subprograms