Chapter 9: Implementing Subprograms

Principles of Programming Languages

Contents

- Implementing “Simple” Subprograms
- Implementing Subprograms with Stack-Dynamic Local Variables
- Nested Subprograms
- Blocks
- Implementing Dynamic Scoping

“Simple” Subprograms

- No nested subprograms
- All local variables are static
  - No recursion

“Simple” Subprograms: Calls and Returns

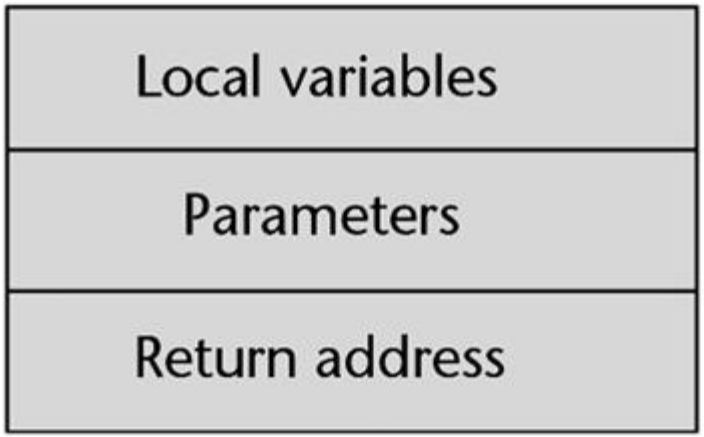
- Subprogram linkage: The subprogram call and return operations of a language
- Actions associated with a subprogram call
  1. Save the execution status of current program unit
  2. Pass the parameters
  3. Pass the return address to the callee
  4. Transfer control to the callee

“Simple” Subprograms: Calls and Returns

- Actions of a subprogram return:
  1. If pass-by-value-result parameters are used, move the current values of those parameters to their corresponding actual parameters
  2. If it is a function, functional value is moved to a place accessible to the caller
  3. Restore the execution status of the caller
  4. Transfer control back to the caller
- What storage does a subprogram **call and return** need?

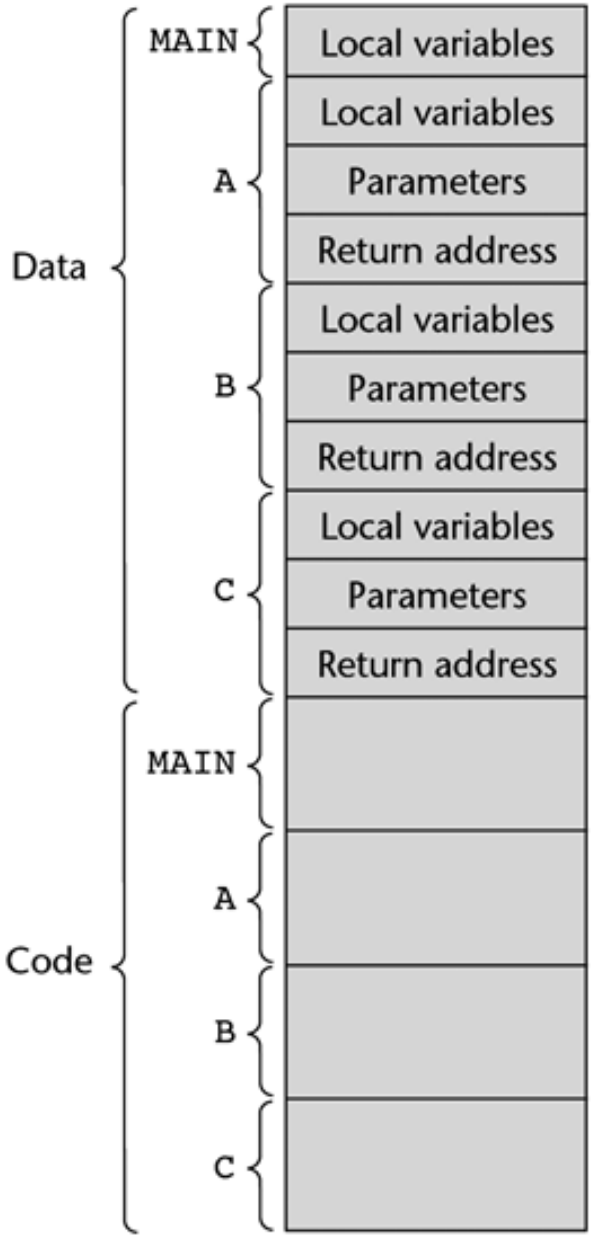
Activation Record

- Activation record: the layout of noncode part of a subprogram
  - Can change when the subprogram is executed
  - Its layout is static



Code and Activation Records

- Can be statically allocated
- Activation records sometimes are attached to their code segment

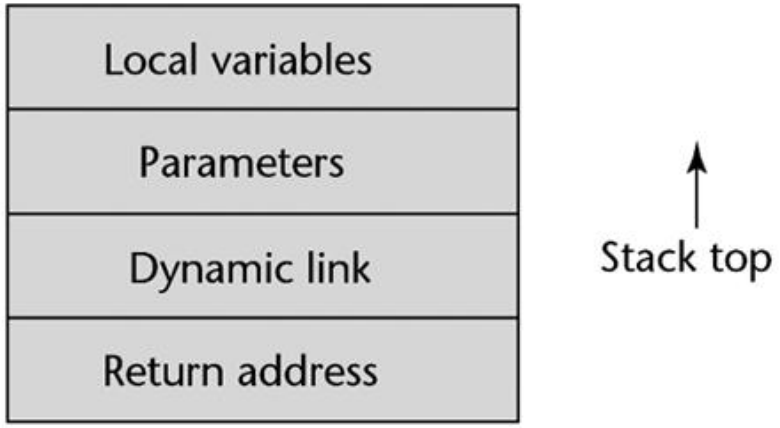


Subprograms with Stack-Dynamic Local Variables

- More complex
  - The compiler must generate code to cause implicit allocation and de-allocation of local variables
  - Recursion must be supported
    - There can be more than one instance of a subprogram at a given time, one from outside call, and one or more recursive calls
    - Each activation requires its own copy of the formal parameters and the dynamically allocated local variables, along with the return address.

New Activation Record

- The activation record format is static, but its size may be dynamic (Ada's array)
- An activation record instance is dynamically created when a subprogram is called
- Some fields are placed by the caller



New Activation Record

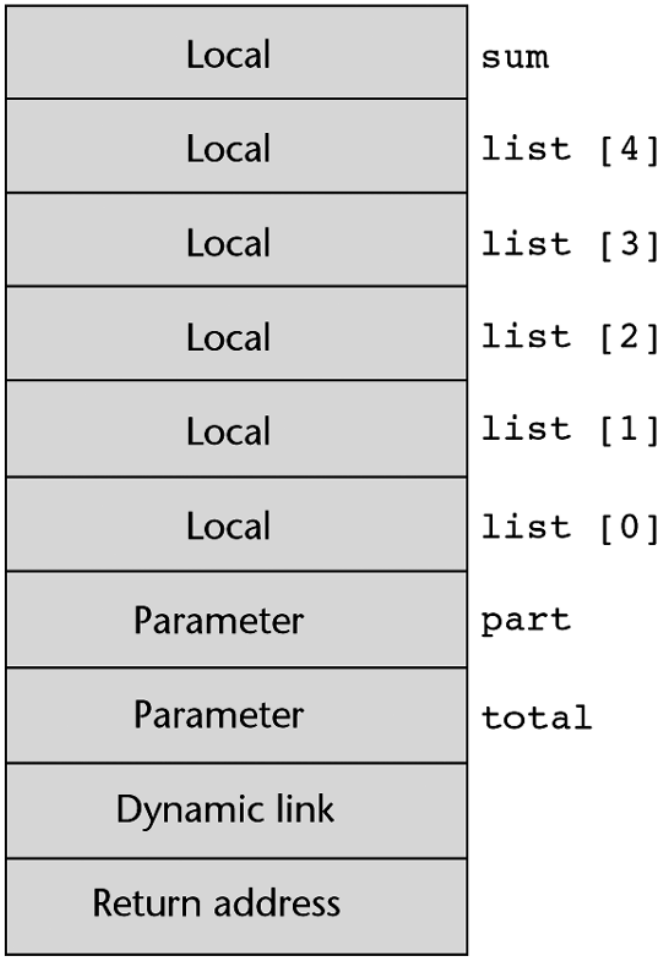
- Return address usually consists of a pointer to the instruction after subprogram call statements
- Dynamic link is the pointer to the top of the activation record instance of the caller
  - Static-scoped languages use this link in destruction of the current activation record
  - The stack top is set to the value of the old dynamic link

New Activation Record

- Parameters are the values or addresses provided by the caller
- Local variables
  - Scalar variables are bound to storage within an activation record instance
  - Structured variables are sometimes allocated elsewhere, only their descriptors and a pointer to that storage are part of activation record

An Example: C Function

```
void sub(float total, int part)
{
   int list[4];
   float sum;
   . . .
}
```

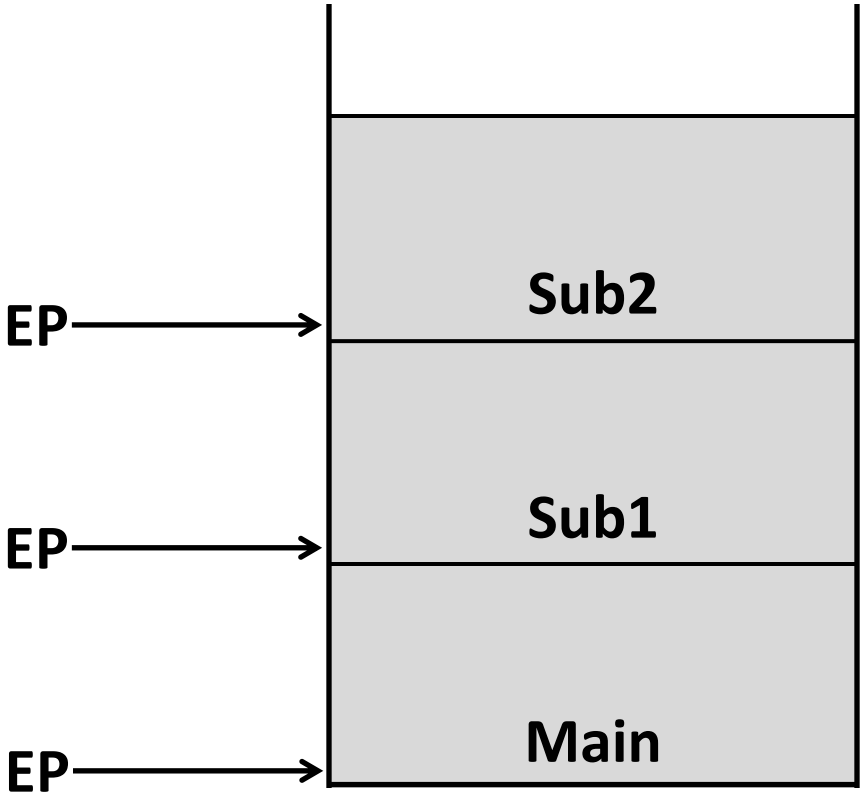


Run-Time Stack

- Subprogram last called is the first to complete
- Create instances of these activation records on a stack: **run-time stack**
- **Environment pointer** (EP) is required to access parameters and local variables during the execution of a subprogram

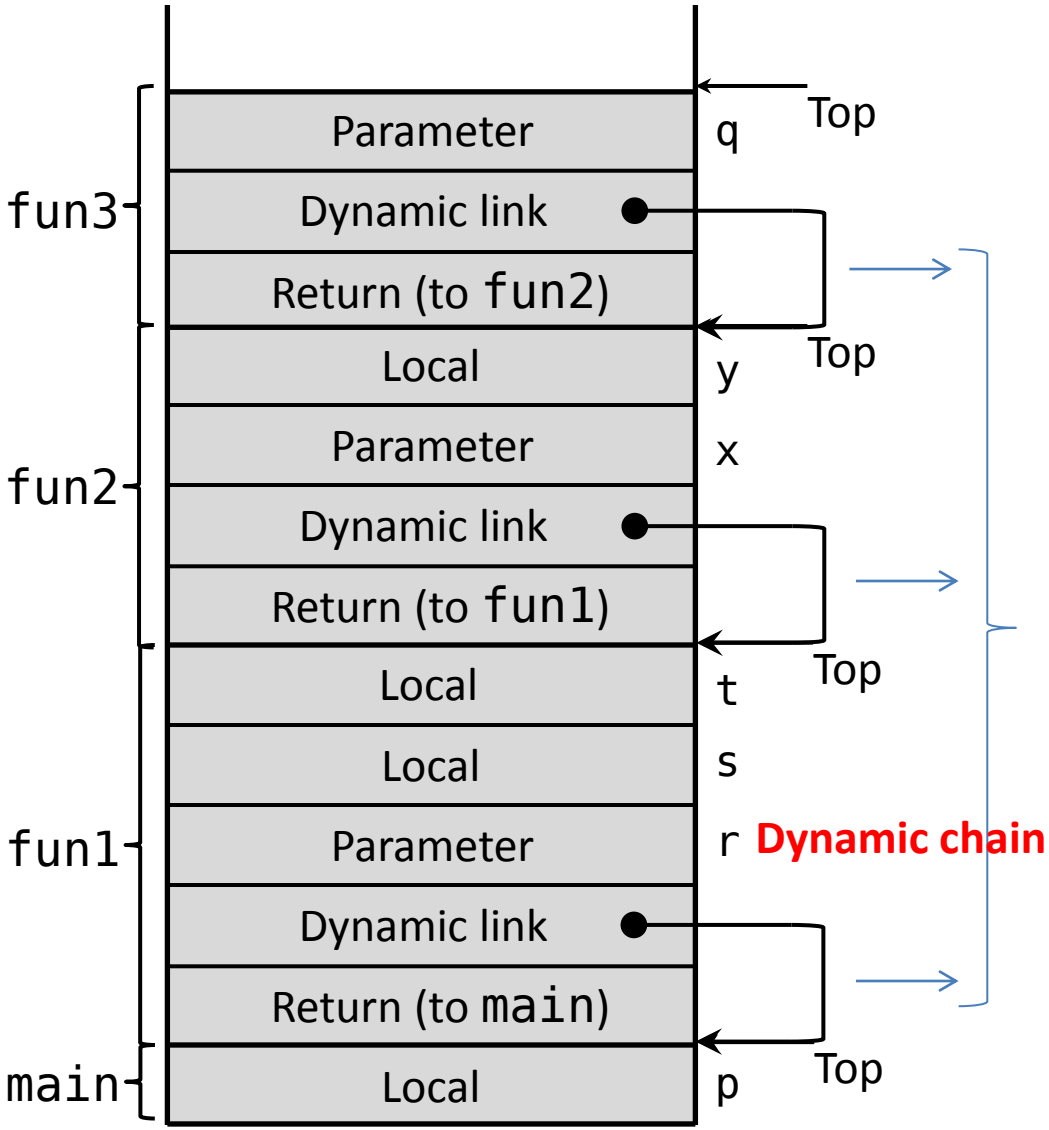
Environment Pointer Illustration

- Only saved versions of EP are stored in the activation record instances
- Saved versions are stored with execution status information



An Example Without Recursion

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



Dynamic Chain and Local Offset

- The collection of dynamic links in the stack at a given time is called the *dynamic chain*, or *call chain*
- Local variables can be accessed by their offset from the beginning of the activation record. This offset is called the *local_offset*
- The local_offset of a local variable can be determined by the compiler at compile time

An Example With Recursion

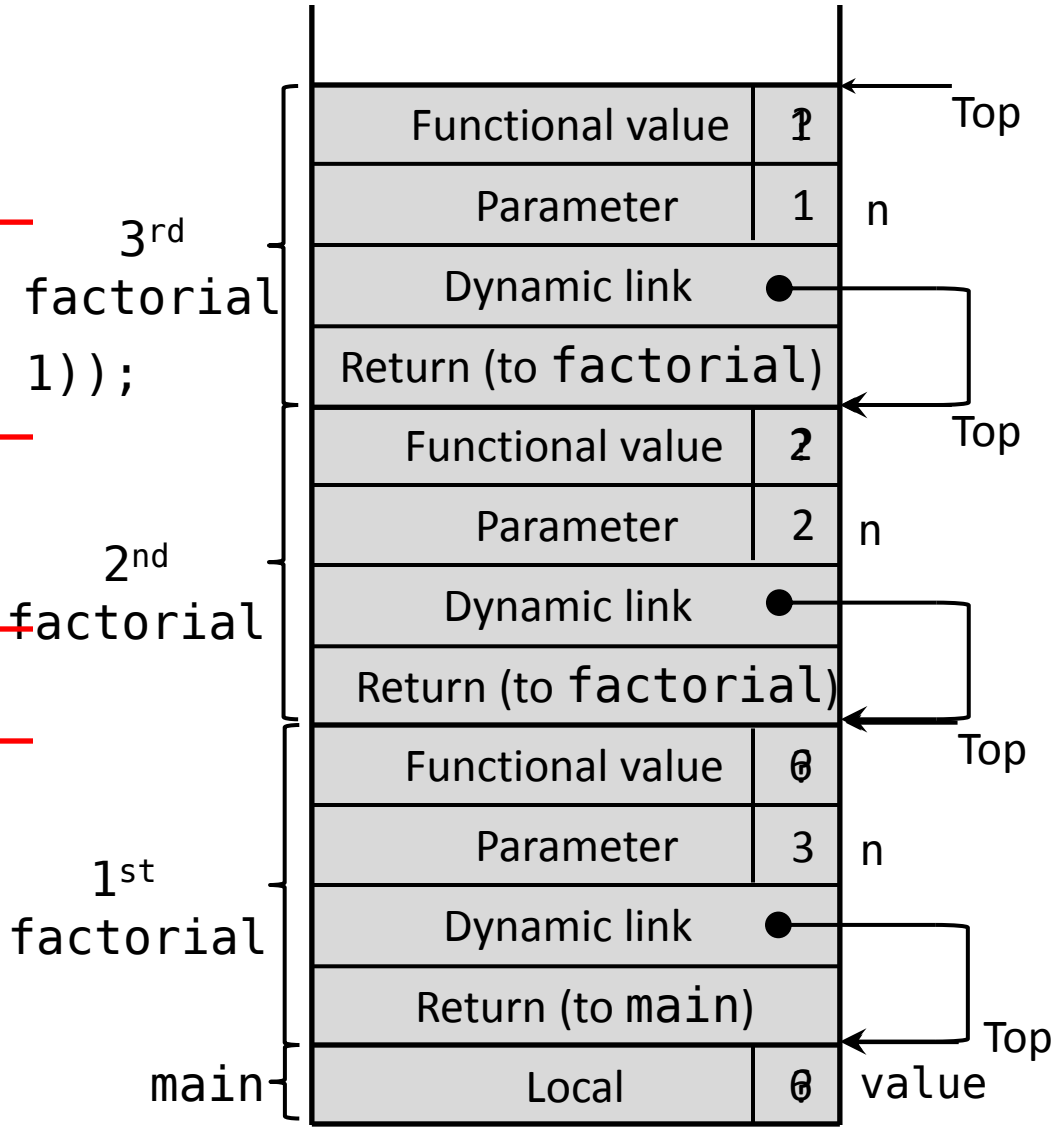
```
int factorial (int n) {

  if (n <= 1) return 1;
    else return
      (n * factorial(n - 1));

}
void main() {
  int value;

  value = factorial(3);

}
```



Nested Subprograms

- Some non-C-based static-scoped languages use stack-dynamic local variables and allow subprograms to be nested
  - Fortran 95, Ada, Python, JavaScript
- All variables that can be non-locally accessed reside in some activation record instance in the stack
- The process of locating a non-local reference:
  1. Find the correct activation record instance
  2. Determine the correct offset within that activation record instance

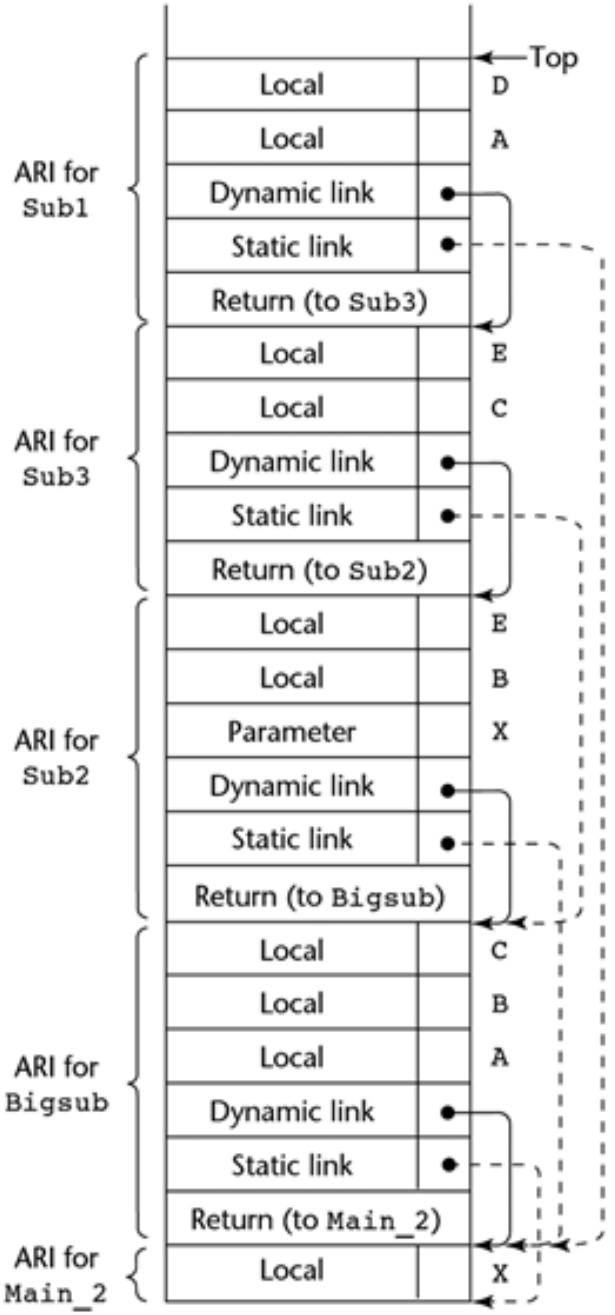
Locating a Non-local Reference

- Finding the offset is easy
- Finding the correct activation record instance
  - Static semantic rules guarantee that all non-local variables that can be referenced have been allocated in some activation record instance that is on the stack when the reference is made

Static Chains

- A new pointer, static link, is added to the activation record
- Static link point from an activation record to the activation record of its static parent
- Used for accesses to nonlocal variables
- A static chain is a chain of static links that connects certain activation record instances
- The static chain from an activation record instance connects it to all of its static ancestors
- (chain_offset, local_offset)

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



Static Chain Maintenance

- Subprogram return: no problem
- Subprogram call:
  - Consider subprogram declaration as variable declaration so that “who declares who” is known statically
  - Calculate the nested_depth from caller to callee’s “father”
  - Trace nested_depth from caller, then connect with callee

Blocks

- Blocks are user-specified local scopes for variables
- C:

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

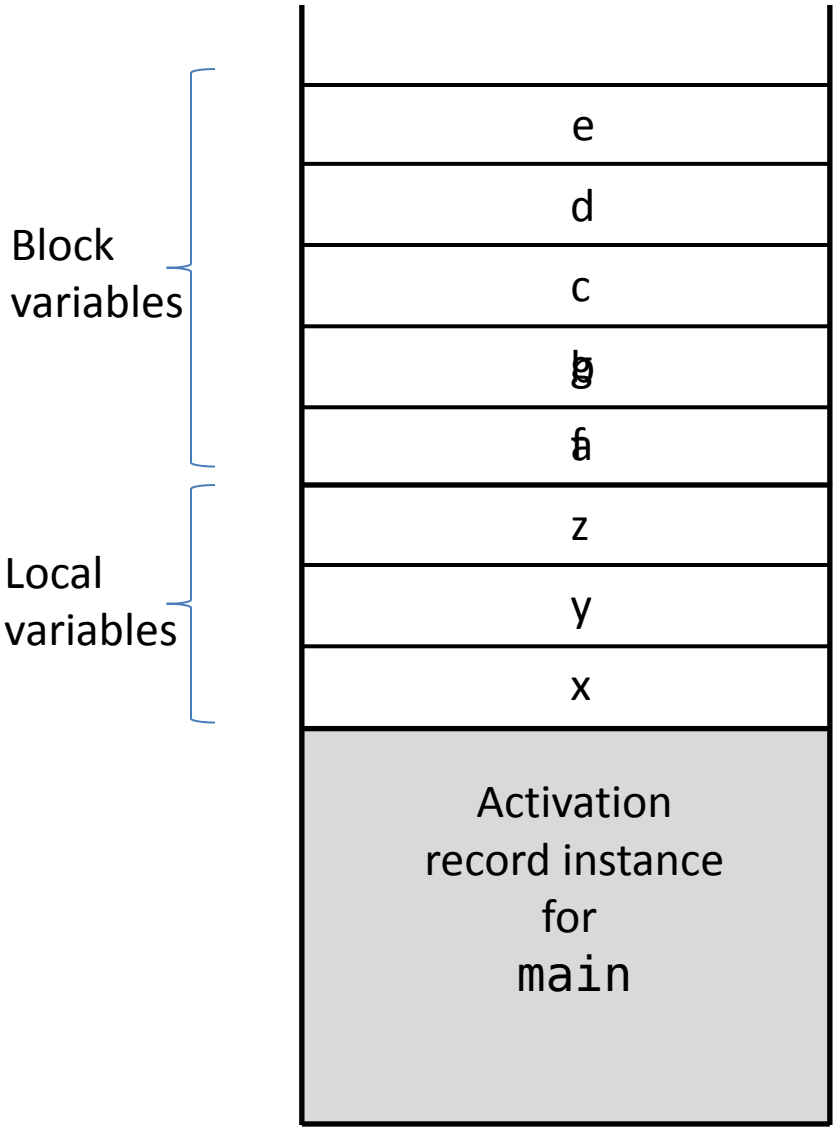
- The lifetime of `temp` in the above example begins when control enters the block and ends when exits
- An advantage of using a local variable is that it cannot interfere with any other variable with the same name

Implementing Blocks

1. Treat blocks as parameter-less subprograms that are always called from the same location
   - Every block has an activation record; an instance is created every time the block is executed
   - Then, static-chain process is used
2. Since the maximum storage required for a block can be statically determined, this amount of space can be allocated after the local variables in the activation record

Implementing Blocks

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

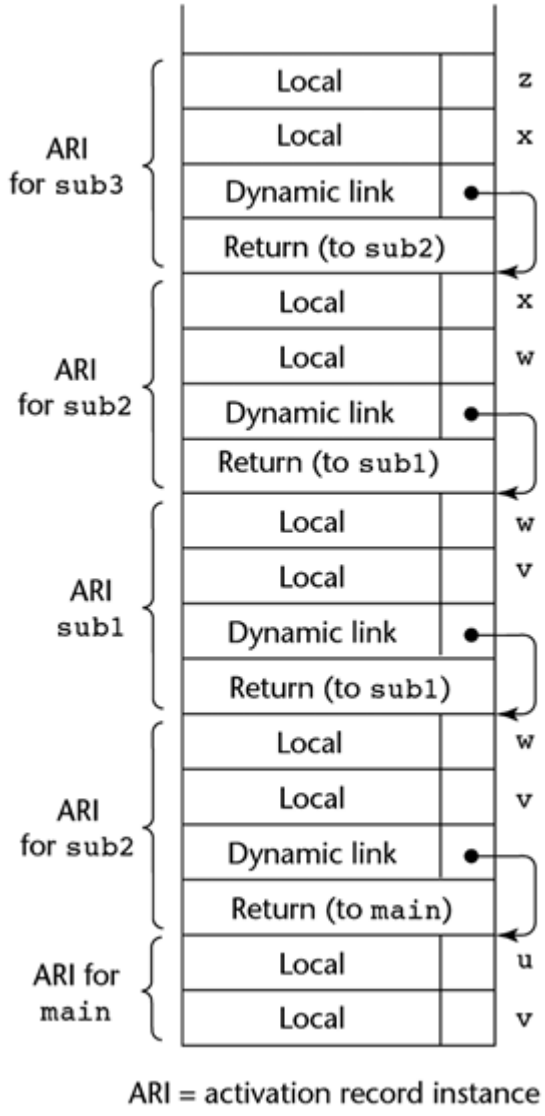


Implementing Dynamic Scoping

- There are two distinct ways: deep access and shallow access
- Don’t be confused with deep and shallow binding in Chapter 8
- *Deep Access*: non-local references are found by searching the activation record instances on the dynamic chain

```
void sub3() {
  int x, z;
  x = u + v;
  ...
}
void sub2() {
  int w, x;
  ...
}
void sub1() {
  int v, w;
  ...
}
void main() {
  int v, u;
  ...
}
```

main → sub2 → sub1 → sub2 → sub3



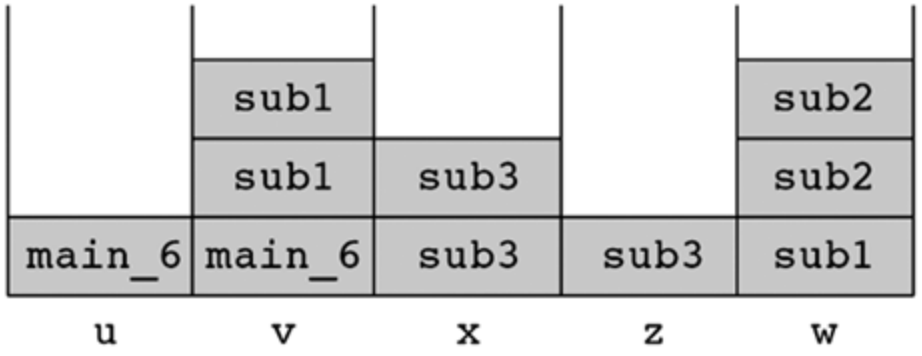
ARI = activation record instance

Shallow Access

- *Shallow Access*: First option
  - Local variables are not stored in the activation records of those program
  - There is a central table with many stacks
  - One stack for each variable name
- Other options for central table:
  1. One stack stores all saved objects, the top one will be accessible
  2. Variables are stored in activation records, but the latest value is stored in a single cells central table

```
void sub3() {
  int x, z;
  x = u + v;
  ...
}
void sub2() {
  int w, x;
  ...
}
void sub1() {
  int v, w;
  ...
}
void main() {
  int v, u;
  ...
}
```

main → sub2 → sub1 → sub2 → sub3



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

Summary (continued)

- 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
- Access to non-local variables in dynamic-scoped languages can be implemented by use of the dynamic chain or thru some central variable table method