Review

• In Algol-like languages, the collection of environments that exist at any point during a computation is embedded in the machine control stack supporting (recursive) procedure calls.
• When the frames of the control stack are used in this way, they are called activation records.
• In each activation record, a pointer called the static link points to the environment parent of the record. Similarly, a pointer called the dynamic link points to the preceding stack frame (activation record).
• The static link is used for looking up bindings in the environment.
• The dynamic link is used to return control from the current procedure to its caller (whose local variables may not be accessible from the current frame).
Consider the following Scheme program to reverse a list:

```
(define rev (lambda (l)
  (letrec
    [(revhelp ; :=
      (lambda (tl acc)
        (if (empty? tl) acc
          (revhelp (rest tl) (cons (first tl) acc)))))]
    (revhelp l empty)))
```

Pidgin Algol equivalent (extended to include functional lists as built-in type):

```
List rev(l: List) = {
  { List revhelp(tl: List, acc: List) = {
    if empty?(tl) then acc else revhelp(rest(tl), cons(first(tl), acc)) }; revhelp(l, empty) }
}
```

What happens when `(rev '(0 1))` is called?

- Top level call on `rev` allocates activation record (AR) #1 with null static and dynamic links and a slot for `l` initialized to `'(0 1)`.
- Body of `rev` (executing in AR #1) allocates AR #2 for block with static and dynamic links pointing to preceding activation record and a slot for `revhelp` initialized to closure for its definition.
- Body of block in `rev` allocates AR #3 record for call on `revhelp` with static link taken from closure binding of `revhelp` and dynamic link pointing to preceding activation record.
Example I cont.

- Since \( l \) is not empty, body of \texttt{revhelp} allocates AR #4 for the recursive call on \texttt{revhelp} with static link taken from closure binding of \texttt{revhelp}, dynamic link …, and slots for \texttt{tl} and \texttt{acc} initialized to \'(1) and \'(0), respectively.
- Since \( l \) is not empty, body of \texttt{revhelp} allocates AR #5 record for recursive call on \texttt{revhelp} with static link taken from closure binding of \texttt{revhelp}, dynamic link …, and slots for \texttt{tl} and \texttt{acc} initialized to \'(())' and \'(1 0)', respectively.
- Since \( l \) is empty, body of \texttt{revhelp} in context of AR #5 returns the value \'(1 0)', popping AR #5 off the stack.
- The pending evaluation in AR #4 returns the value \'(1 0)', popping AR #4.
- The pending evaluation in AR #3 returns the value \'(1 0)', popping AR #3.
- The pending evaluation in AR #2 returns the value \'(1 0)', popping AR #2.
- The pending evaluation in AR #1 returns the value \'(1 0)', popping AR #1

Notes:
1. The last four steps are trivial because they are returns from tail calls.
2. The dynamic links is \textit{always} set to point to the preceding AR.
3. Algol 60 was designed so that the ARs could be stack allocated (and deallocated). Guy Steele’s heap allocation “hack” (to be covered later) extends the stack allocation protocol to languages supporting first-class functions/procedures. In Java, there are syntactic restrictions that effectively impose the same allocation constraint.
Consider the following Scheme program to reverse a list:

```scheme
(define lookup (lambda (sym env)
  (letrec
    [(lookup-help (lambda (env)
                   (cond [(empty? env) null]
                         [(eq? sym (pair-var (first env))
                           (pair-val (first env))]
                         [else (lookup-help (rest env) tl)]
                         )))
     (lookup-help env)))
```

Let’s trace the evaluation of `(lookup 'a (cons (make-pair 'a 5) null))`:

- The top-level call on `lookup` allocates AR #1 with null static link and slots for `sym` and `env` initialized to `'a` and `'((['a 5]))`.
- The body of `lookup` (executing in AR #1) allocates AR #2 for the block with static link pointing to AR #1 and a slot for `lookup-help` initialized to the closure for its definition.
- The body of the block executing in AR #2 allocates AR #3 for the call on `lookup-help` with the static link extracted from the closure bound to `lookup-help` and a slot for `env` initialized to `'((['a 5]))` (the value of `env` following the static link of AR #2).
Exceptions

- Exceptions were not included in Algol 60 or most of its successors (Pascal, Algol W, C). But the Algol 60 run-time stack can easily handle the Java `try/catch` construct.

- How does exception handling work? The activation record must include a `catch` table for the active `catch` (assuming one exists) listing the caught exception classes (types) and their handlers (the bodies of the `catch` clauses). (A `catch` is active if control is within the corresponding `try` block.)

- When an exception is thrown the executing code (interpreter or compiled code) searches back through the dynamic chain --- popping exited frames off the stack --- to find the first matching `catch` clause.