Objectives

1. be able to describe the components and runtime layout activation records in block structured languages

2. be able to identify which incremental part(s) of the book's reference implementation are necessary to support each of the following features: blocks, nested blocks, functions/procedures, functions with static scope, and functions as arguments and results

3. be able to identify the circumstances in which stack discipline can or cannot be used for activation records

4. be able to draw/evolve the runtime layout of call stacks (with environment pointers and activation records), closures, function code, etc., as applicable, for C and ML programs at different points in their execution assuming the book's reference implementation of procedure calls

5. be able to trace (simulate by hand) the execution of programs in an Algol-, C-, or ML-like programming language with static and dynamic scoping for non-local variables in nested functions.

Outline

1. review static vs. dynamic
   (a) static: when the code is processed (often related to the structure of the source code)
   (b) dynamic: as the code is executed (often related to the execution trace)
   (c) in a compiled language like C
      i. ‘static’ means during program compilation
      ii. ‘dynamic’ means during program execution
   (d) even in a dynamic language like Lisp
      i. creation of function value with ”lambda” = ‘static’ with respect to that function
      ii. calling of the function = dynamic

2. block structure
   (a) nested blocks
   (b) compared to Fortran
   (c) activation records for inline blocks (reference implementation)
      i. control links for nested blocks
      ii. name bindings resolved statically

Reading:

- Mitchell, Ch. 7
iii. each variable accessible through static coordinates
iv. fields: control link, local variables, intermediate results

3. function calls

(a) activation record
   i. call stack is dynamic, distinct from lexical nesting
   ii. fields shared with inline blocks: control link, local variables, intermediate results
   iii. additional fields: parameters, return address, return-result address
(b) parameter passing
   i. pass-by-value
   ii. pass-by-reference
(c) static (lexical) vs. dynamic scoping of variables / parameters
   i. nested first-order functions
   ii. dynamic scoping
   iii. lexical scoping
   iv. access links
(d) tail recursion
(e) higher-order functions
   i. first-class/second-class functions
   ii. passing nested functions parameters (second-class)
      A. ‘downward funarg problem’
      B. closure
   iii. returning function parameters (first-class)
      A. ‘upward funarg problem’
      B. garbage collected activation records (no stack discipline)

Vocabulary

block, local variable, parameter, non-local (global) variable, activation record (= stack frame),
stack discipline, inline block, scope, lifetime, control link (a.k.a. dynamic link), actual parameters,
formal parameter, environment pointer (= frame pointer = $fp frame pointer on MIPS, rbp on AMD64),
access link (a.k.a. static link), first-order functions, higher-order functions,
closure, pass-by-value, pass-by-reference