

Runtime Environments

- Procedure call-return model
- Scope and lifetime of declarations
- Lexical vs. dynamic scoping
- Run-time memory organization
- Activation records

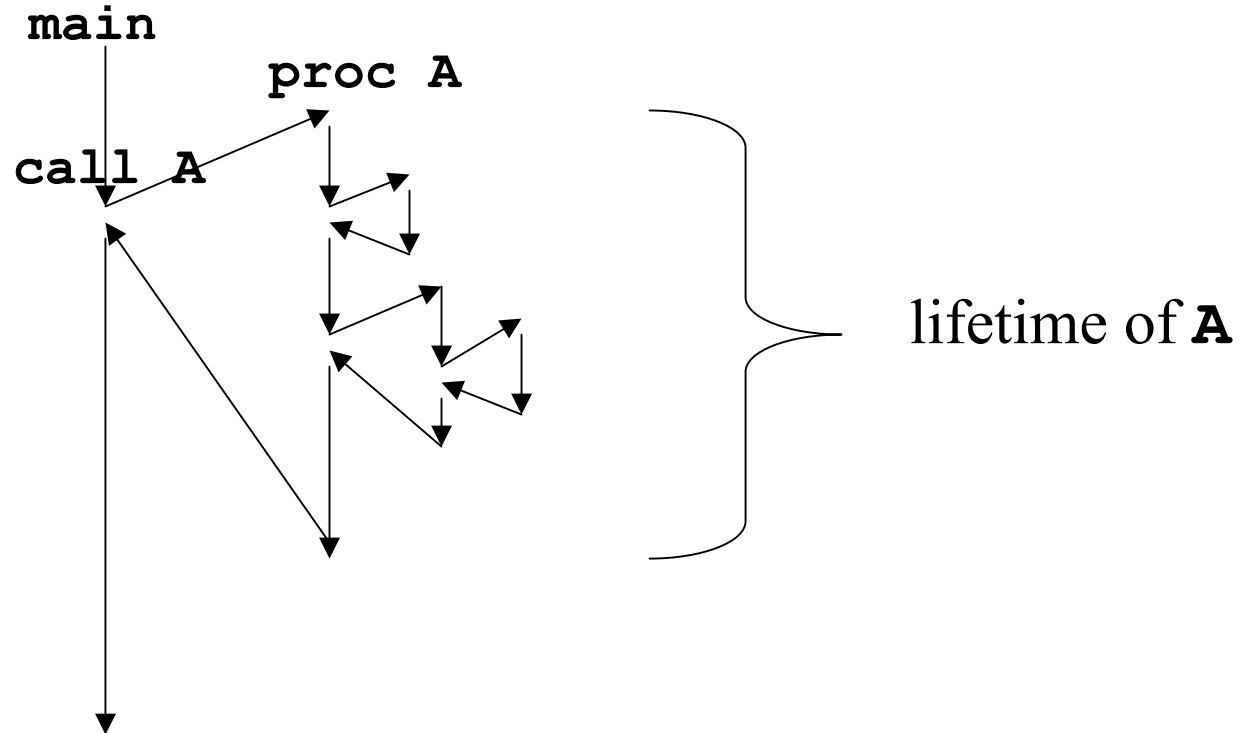
Intro

- To generate code, need to know about environment program will be running in
- Two types of assumptions made by compiler:
 - mapping between names in the prog. and data objects on the machine
 - O.S.-supplied library routines for I/O, exception handling, dynamic storage mgmt., etc.

(this lecture focuses on the former)

Activation and lifetime

- *activation* of a procedure = execution of a proc.
- *lifetime* of an activation of proc **A** is total time control is in **A**, or in procs. called by **A**, until **A** returns.

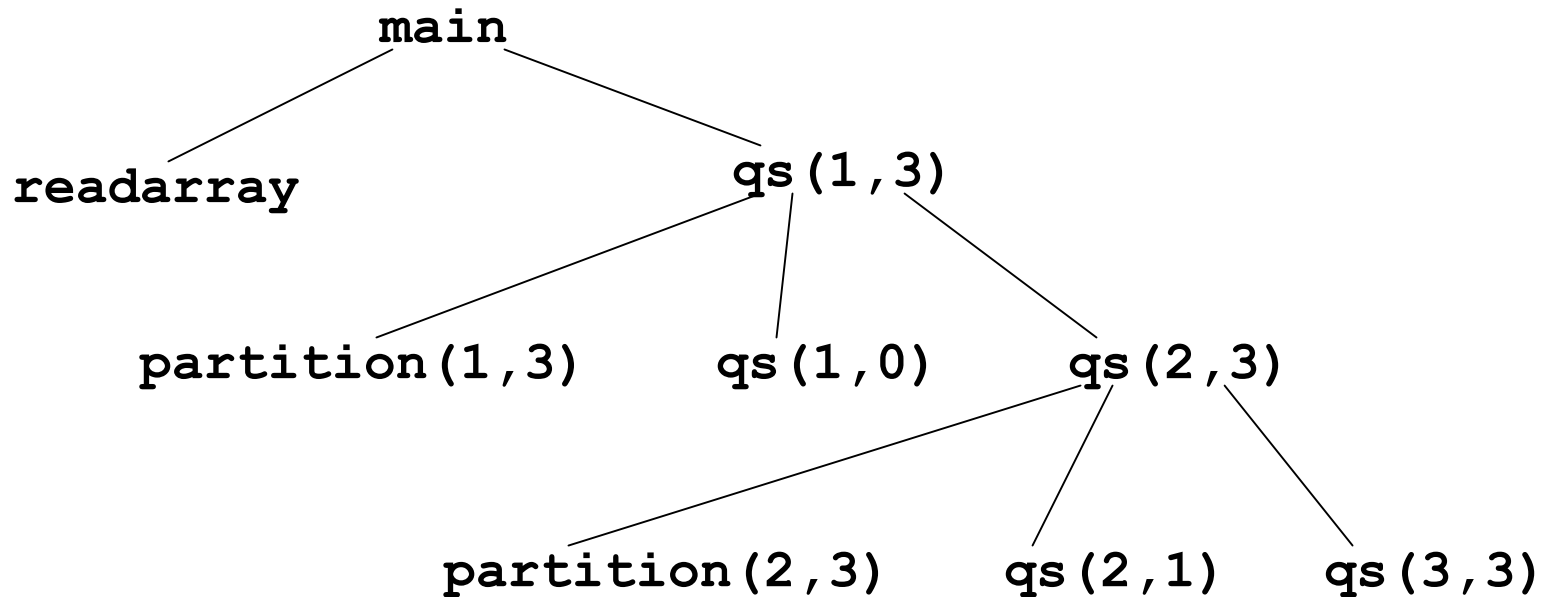


Activation trees

- for a recursive proc. several activations may be alive at the same time
- *activation tree* shows the way control enters and leaves activations for one run of a program.
 - each node represents an activation of a proc.
 - parent \rightarrow child, means parent called child
 - sibling *a* left of sibling *b* means call to *a* happened before call to *b*.
- Example follows...

Example activation tree

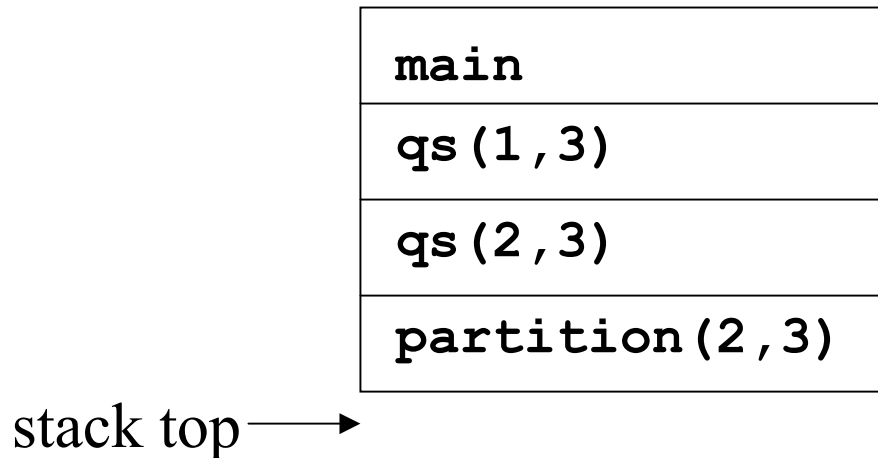
- for a program with a recursive quicksort routine called “qs” (ex. from Aho, Hopcroft and Ullman text)



- at runtime need info about activations on path from root to currently executing leaf

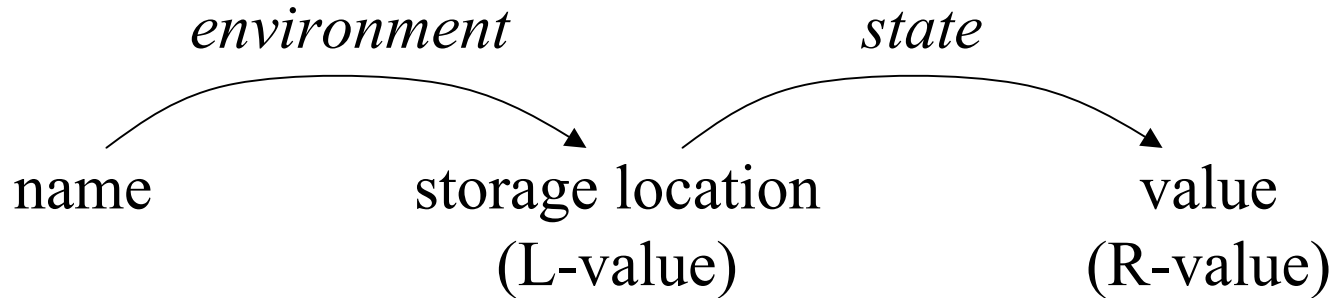
Control stack

- the *control stack* is a data structure used at run-time to keep track of live procedure activations.
(a.k.a., runtime stack, call stack)
- push a node onto the stack at the beginning of lifetime, and pop it at end of lifetime.
- contents when control is in **partition (2, 3)**:



Environments

- *scope* rules determine which declaration of a name applies



- a declaration changes the environment: `int x;`
- an assignment changes the state: `x = 23;`
- L-value: name denotes where a value is to be stored
- R-value: name denotes the value stored at the location
 - (from LHS and RHS of assgt. E.g., `i = i + 1;`)

Bindings and lifetime

- an association between a name, **x**, and a storage location, **s**, in an environment is called a *binding* of **x** to **s**.
- *lifetime* (or *extent*) of a variable is the portion of the execution where an instance of x is usable (binding valid). C language example:
 - automatic local: local scope, lifetime is lifetime of procedure activation
 - static local: local scope, but lifetime of whole program
- with recursion can have more than one binding of a var in effect at the same time (store these locs on the control stack)

Lexical vs. dynamic scoping

- lexical (or static scoping):
 - which declaration applies determined at compile-time
 - innermost-enclosing-block rule
 - name resolution independent of caller
 - used by most modern languages: C/C++, Pascal, etc.
- dynamic scoping (no longer used)
 - which declaration applies determined at run-time
 - most-recent-occurrence rule
 - name resolution depends on caller
 - formerly used in some interpreted languages (older versions of lisp)

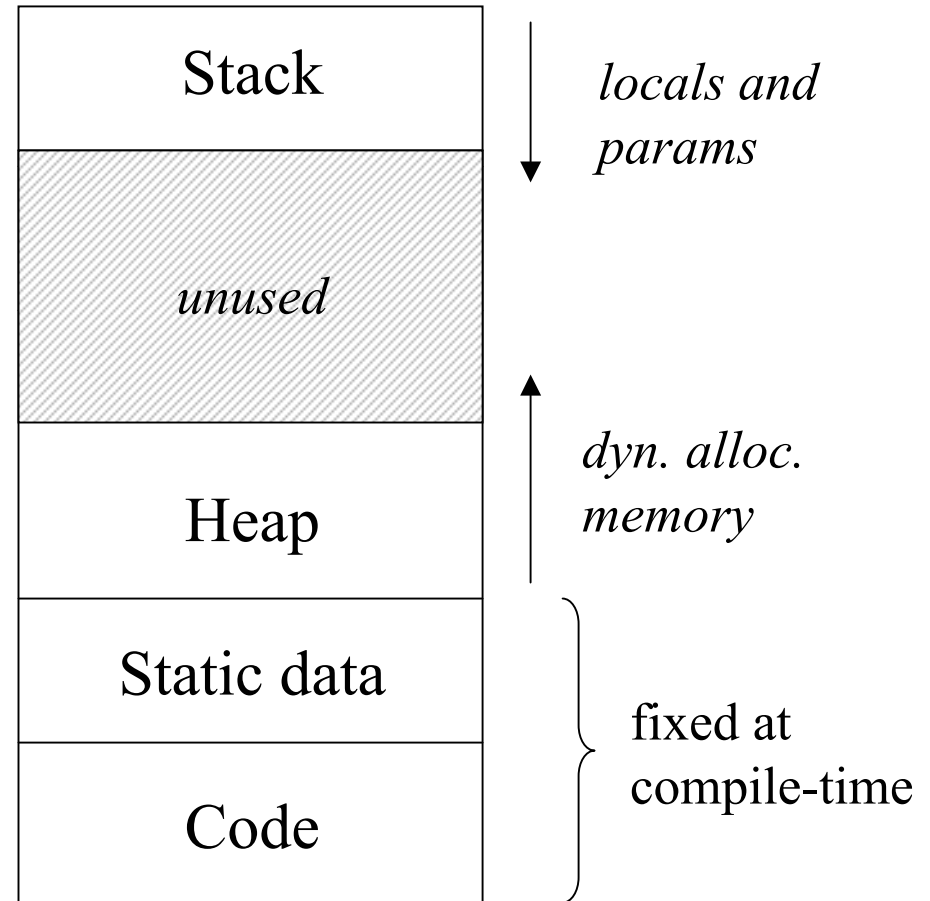
Where do the variables go in memory?

- Earliest programming languages used static storage allocation
 - all addresses (bindings) computed at compile-time
 - ok for traditional Fortran, Basic:
 - no recursion, no dynamic data structures
 - compiler keeps mapping of bindings (in symbol table)
 - without recursion we can only have one binding of a variable at one time.
 - all activations can use the same binding.
 - no need for control stack
- Modern programming languages use dynamic alloc. . . .

Dynamic storage allocation

- to accommodate recursion:
 - allocate space for params and locals at run-time on every proc. activation
 - dealloc. on proc exit
- to accommodate dynamic data structures:
 - dynamic allocation as you go (via “malloc” or “new”)
 - programmer can dealloc, or garbage collection can be done

Common memory organization:



Note: Diagram shows MIPS memory layout

Activation records

- The block of data pushed on the control stack for one proc. activation is called an *activation record* (or *stack frame*)
 - corresponds to one “node” in the activation tree
 - contains all info needed for this activation, and to resume activation of the caller successfully.
 - anything that can be overwritten by each procedure activation must be saved in an activation record
 - total size can be determined at compile-time, but activation records for different procs. will be different sizes (e.g., because they have different numbers of locals)
 - let’s see what’s in it . . .

Activation record contents

- returned value
- actual parameters
- access link (opt.)
 - pointer to a.r. of enclosing scope [for langs with nested funcs]
- control link (opt.)
 - pointer to a.r. of caller
- saved machine status
 - save register values, incl. PC
- local variables
- temporaries
 - used in evaluating expressions