Procedure Activation and Lifetime

• A procedure is *activated* when called
• The *lifetime* of an activation of a procedure is the sequence of steps between the first and last steps in the execution of the procedure body
• A procedure is *recursive* if a new activation can begin before an earlier activation of the same procedure has ended
Program sort(input, output)
  var a : array [0..10] of integer;
procedure readarray;
  var i : integer;
  begin
    for i := 1 to 9 do read(a[i])
  end;
function partition(y, z : integer) : integer
  var i, j, x, v : integer;
  begin ...
  end
procedure quicksort(m, n : integer);
  var i : integer;
  begin
    if (n > m) then begin
      i := partition(m, n);
      quicksort(m, i - 1);
      quicksort(i + 1, n)
    end
  end
begin
  a[0] := -9999; a[10] := 9999;
  readarray;
  quicksort(1, 9)
end.

Activations:
begin sort
  enter readarray
  leave readarray
  enter quicksort(1,9)
  enter partition(1,9)
  leave partition(1,9)
  enter quicksort(1,3)
  ...  
  leave quicksort(1,3)
  enter quicksort(5,9)
  ...  
  leave quicksort(5,9)
  leave quicksort(1,9)
end sort.
Activation Trees: Example

Activation tree for the sort program
Note: also referred to as the dynamic call graph
Control Stack

Activation tree:

Control stack:

Activations:

```
begin sort
enter readarray
leave readarray
enter quicksort(1,9)
enter partition(1,9)
leave partition(1,9)
enter quicksort(1,3)
enter partition(1,3)
leave partition(1,3)
enter quicksort(1,0)
leave quicksort(1,0)
enter quicksort(2,3)
...```
Scope Rules

- *Environment* determines name-to-object bindings: which objects are in *scope*?

```plaintext
program prg;
    var y : real;
function x(a : real) : real;
    begin … end;
procedure p;
    var x : integer;
    begin
        x := 1;
        …
        end;
begin
    y := x(0.0);
    …
end.
```

Variable `x` locally declared in `p`

A function `x`
Mapping Names to Values

```plaintext
var i;
...
i := 0;
...
i := i + 1;
```
Mapping Names to Values

At compile time

\textit{environment}

name \quad storage

At run time

\textit{state}

value

\textbf{var} \ i; \\
\ldots \\
i \ := \ 0; \\
\ldots \\
i \ := \ i + 1;
# Static and Dynamic Notions of Bindings

<table>
<thead>
<tr>
<th>Static Notion</th>
<th>Dynamic Notion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition of a procedure</td>
<td>Activations of the procedure</td>
</tr>
<tr>
<td>Declaration of a name</td>
<td>Bindings of the name</td>
</tr>
<tr>
<td>Scope of a declaration</td>
<td>Lifetime of a binding</td>
</tr>
</tbody>
</table>
Stack Allocation

- *Activation records* (subroutine frames) on the runtime stack hold the state of a subroutine
- *Calling sequences* are code statements to create activations records on the stack and enter data in them
  - Caller’s calling sequence enters actual arguments, control link, access link, and saved machine state
  - Callee’s calling sequence initializes local data
  - Callee’s return sequence enters return value
  - Caller’s return sequence removes activation record
Activation Records
(Subroutine Frames)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Returned value</td>
<td></td>
</tr>
<tr>
<td>Actual parameters</td>
<td></td>
</tr>
<tr>
<td>Optional control link</td>
<td></td>
</tr>
<tr>
<td>Optional access link</td>
<td></td>
</tr>
<tr>
<td>Saved machine status (w/ opt. return address)</td>
<td></td>
</tr>
<tr>
<td>Local data</td>
<td></td>
</tr>
<tr>
<td>Temporaries</td>
<td></td>
</tr>
</tbody>
</table>

(fp) (frame pointer)

- Caller’s responsibility to initialize
- Callee’s responsibility to initialize
Control Links

The control link is the old value of the fp

fp → Control link

sp → Stack growth

Caller’ s activation record

Callee’ s activation record
program sort(input, output)
    var a : array [0..10] of integer;
    x : integer;
procedure readarray;
    var i : integer;
begin ... end;
procedure exchange(i, j : integer);
    begin x := a[i]; a[i] := a[j]; a[j] := x end;
procedure quicksort(m, n : integer);
    var k, v : integer;
    function partition(y, z : integer) : integer
        var i, j : integer;
        begin ... exchange(i, j) ... end
    begin
        if (n > m) then begin
            i := partition(m, n);
            quicksort(m, i - 1);
            quicksort(i + 1, n)
        end
    end;
begin
    ... 
    quicksort(1, 9) 
end.
Access Links (Static Links)

The access link points to the activation record of the static parent procedure:
s is parent of r, e, and q
q is parent of p
Accessing Nonlocal Data

• To implement access to nonlocal data $a$ in procedure $p$, the compiler generates code to traverse $n_p - n_a$ access links to reach the activation record where $a$ resides
  – $n_p$ is the nesting depth of procedure $p$
  – $n_a$ is the nesting depth of the procedure containing $a$
Parameter Passing Modes

- **Call-by-value**: evaluate actual parameters and enter r-values in activation record
- **Call-by-reference**: enter pointer to the storage of the actual parameter
- **Copy-restore** (aka value-result): evaluate actual parameters and enter r-values, after the call copy r-values of formal parameters into actuals
- **Call-by-name**: use a form of in-line code expansion (*thunk*) to evaluate parameters