

CS 4100

Block Structured Languages

April 13, 2011
Based on slides by
Istvan Jonyer
Book by
MacLennan

1

Chapter 6: Implementation of Block-Structure

- Addressing implementation aspects of block-structured languages (Pascal and Algol)
 - Fortran (and pseudocode) not block structured
 - We'll focus on Pascal, since most languages these days are Pascal-like
 - Algol is block structured

2

Activation Record

- Represents the state of a procedure

3

Fixed vs Variable

- Program has two major components
 - Fixed part
 - Code (the program itself)
 - Does not change during runtime
 - Variable part
 - Activation record
 - Dynamically created and deleted at runtime
 - We'll focus on this part

4

State of an Activation

- Point of execution (instruction pointer)
 - Stored in IP of activation record (and IP register of processor)
 - Usually points to next instruction
- Context of execution (scope/environment)
 - Environment pointer (EP)
 - Local context
 - Local activation record
 - Non-local context
 - Non-local activation record

5

Activation Records

- Local variables and formal parameters are contained in the activation record
 - Create and delete correspond to entry and exit
- Context of a statement
 - Names declared in current procedure +
 - Names declared in surrounding procedures
 - For multiple bindings, innermost declaration is used (if name not found in current activation record, look to outer A/Rs successively)

6

Static Link

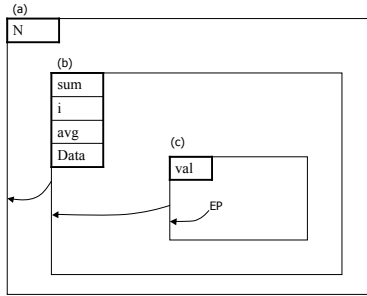
- How to keep track of outer scopes? (p214)
 - Static link points to outer activation record
 - Each context (A/R) has static link to outer scope
 - Static links form a chain all the way to top level (global scope, and beyond to OS)
- Static chain reflects the static structure of the program
 - The way procedures are nested
 - Ends at global scope

7

```
program a (...);
  var N: integer;
  procedure b(sum: real);
    var i: integer;
        avg: real;
        Data: array[1..10] of real;
    procedure c(val: real);
    begin
      writeln (Data[i]);
    end; // c
  begin // b
    ...
  end; // b
begin // a
  ...
end; // a
```

8

Contour Diagram of Static Structure of Previous Program



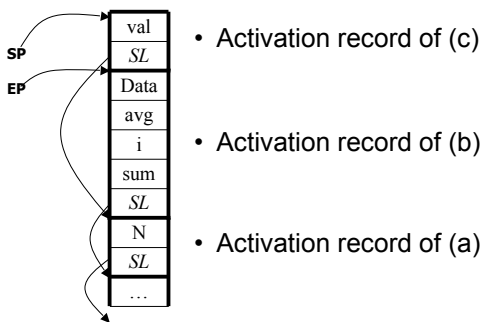
9

Pointers

- EP points to active local context
- SP points to register of active context
- IP points to next instruction
- SL outer activation record
 - Environment of declaration
 - Keeps track of outer scopes
- DL (coming soon) points to callers A/R
 - Talked about this in Fortran

10

Activation Record for Procedures



11

Variable Addressing

- Name lookup is done at compile time
 - Names are not actually looked up at runtime
 - Names are bound to addresses in activation record
- We need two addresses for accessing a variable
 - How far we have to follow the static link
 - Where the variable is defined
 - Offset of variable in activation record

12

Terminology

- Static nesting level
 - How deep the scope is where variable is defined (from global scope)
 - Number of contour lines surrounding declaration or use
- Static distance
 - Distance between the variable's declaration and use
- Offset
 - Variables position inside activation record

13

Fetching a Variable

- Notation
 - $M[i]$: memory at address i
 - EP: environment pointer (how to get to A/R)
 - $\text{offset}(v)$: relative offset of variable v in activation record (how to find in A/R)
 - reg.X : processor register (EP,IP,SP)
- General case (v is local)
 - `fetch` $M[\text{reg.EP} + \text{offset}(v)]$

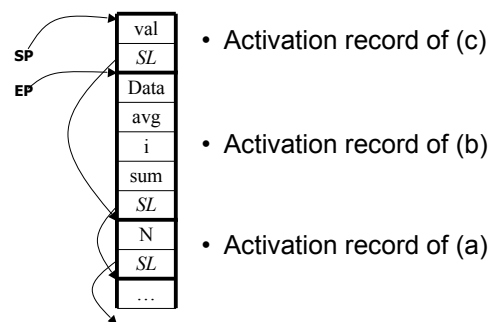
14

Examples

- Get variable `sum` (with offset 1) at static distance of 1
 - ARP: activation record pointer
 - `ARP := M[reg.EP];`
 - `fetch M[ARP + 1];`
- Get variable `N` (with offset 1) at static distance of 2
 - `ARP := M[reg.EP];`
 - `ARP := M[ARP];`
 - `fetch M[ARP + 1];`

15

Activation Record for Procedures



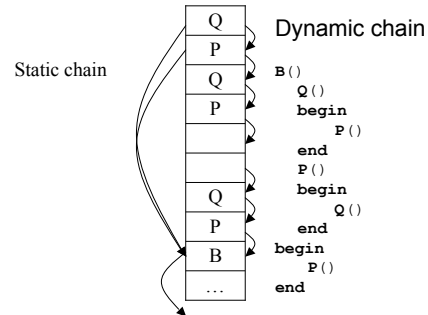
16

Dynamic Link

- How is dynamic link different from static link?
 - Can we do with just one?
 - Both are needed for static scoping
 - Dynamic link is enough for dynamic scoping
- Static link
 - Points to environment of declaration
- Dynamic link
 - Points to caller
 - Can restore caller's state on exit

17

Dynamic vs Static Link



18

Procedure Activation

- Three steps
 - Save state of caller
 - In local activation record
 - Create activation record of callee
 - Transmit parameters to callee
 - Establish dynamic link from caller
 - Enter callee
 - At its first instruction

19

Saving the Caller's State

- Saving address where caller must resume after returning from call
- Saving locals and non-locals
 - No action is required
 - Locals are already stored in AR
 - Access to non-locals is already established (SL)
- Saving processor registers
 - Registers must be saved in AR
 - Platform-specific (not discussed)
 - Not visible to programmer

20

Creating Callee's AR

- Callee's AR has following components
 - PAR: parameters
 - Parameters are placed here by caller
 - $M[\text{callee's AR}].\text{PAR}[1] := \text{evaluation of parameter 1};$
 - IP: resumption address
 - Not used until making procedure call
 - SL: static link
 - Set to environment of definition
 - Computed from static nesting levels of procedures
 - $M[\text{callee's AR}].\text{SL} := \text{reg.EP}$ (if defined in current scope)
 - DL: dynamic link
 - Set to caller's AR (EP register)
 - $M[\text{callee's AR}].\text{DL} := \text{reg.EP}$

21

Final Steps

- Install callee's AR as current activation record
 - $\text{reg.EP} := \text{callee's AR};$
- Include callee's AR in stack "officially"
 - $\text{reg.SP} := \text{reg.SP} + \text{size}(\text{callee's AR});$
 - `goto entry(callee);`
- Both entry point and AR size are known at compile time
 - $\text{Goto} = \text{reg.IP} := \text{entry}(\text{callee})$

22

Procedure Exit

- We have to effectively reverse the entry procedure
 - Delete callee's activation record
 - Subtract size of AR from stack
 - $\text{reg.SP} := \text{reg.SP} - \text{size}(\text{callee's AR})$
 - Restore the state of the caller
 - Reinstalling the caller's context
 - $\text{reg.EP} := M[\text{reg.EP}].\text{DL};$
 - Resume execution of caller
 - $\text{reg.IP} := M[\text{reg.EP}].\text{IP}$ (`goto M[reg.EP].IP`)

23

Non-Local GOTOS

- Local GOTO
 - Simple machine jump to address
- Non-local GOTO
 - Requires restoration of environment
 - Must manipulate runtime stack
 - Analogous to returning from a procedure call

24

Example

```

B()
  Q()
    P()
      begin
        ...
        goto 1;
      end
    begin
      P()
        ...
      end
    begin
      Q()
1: ...
    end
  end
end

```

25

Implementation

- How do we find the scope for the label?
 - Static nesting level is kept in symbol table at compile time
 - Static difference *sd* can be computed and found runtime
- Steps involved:
 - Scan down static chain *sd* times
 - $sd \text{ times: } \text{reg.EP} := M[\text{reg.EP}].\text{SL}$
 - Remove ARs from top of stack
 - $\text{reg.SP} := \text{reg.SP} + \text{size}(\text{AR of label})$
 - Transfer execution to point of label (constant)
 - $\text{goto address}(\text{label})$

26

Displays

- Traversing static chains is proportional to length of chain
 - Would be nice if it was constant
- Solution
 - Store the address of activation record for each environment (not procedure call!) in array
 - This array is called the "display" *D*
 - Accessing static nesting levels is easy
 - $D[\text{sn}]$
 - Accessing variables is now only two steps, always!
 - $\text{fetch } M[D[\text{sn}] + \text{offset}(\text{variable})]$

27

Static Chains vs. Displays

| Operation | Static Chain | Display |
|--------------------|--------------|---------|
| Local variable | 1 | 2 |
| Non-local variable | $sd+1$ | 2 |
| Procedure call | $sd+3$ | 6 |
| Procedure return | 2 | 5 |

- SC values are estimates

- Displays

- Better for variables
- Worse for procedure calls

28

Blocks

- Pascal does not have blocks...
- But Algol, C, Ada and many others do
- Blocks require activation records
 - Thus, entering and exiting a block is analogous with calling and returning from a procedure
 - Can they be implemented in the same way?
 - Yes!

29

Block vs. Procedure

- Some efficiency hacks are possible with blocks
 - Blocks are always called from the same place! ...and returns to the same place!
 - No need to save IP (resume address) of caller
 - No need to save processor registers
 - Environment is always the same
 - Environment of definition = Surrounding block
 - Static and dynamic links are the same
 - No parameters
 - No need to evaluate and copy parameters

30

Improvements

- Simplified structure
 - LV: local variables
 - Block can have local variables
 - (vs. compound statements)
 - IP: resumption address
 - Block may call procedure
 - SL: static link
 - Remove dynamic link, since they are the same

31

Entry-Exit

- Entry:

```
M[reg.SP].SL := reg.EP;
reg.EP := reg.SP;
reg.SP := reg.SP + size(block AR)
```
- Exit

```
reg.SP := reg.SP - size(block AR)
reg.EP := M[reg.EP].SL;
```

32