Contour Model for procedure activation

Peter J. Denning

© 2019, Peter J. Denning

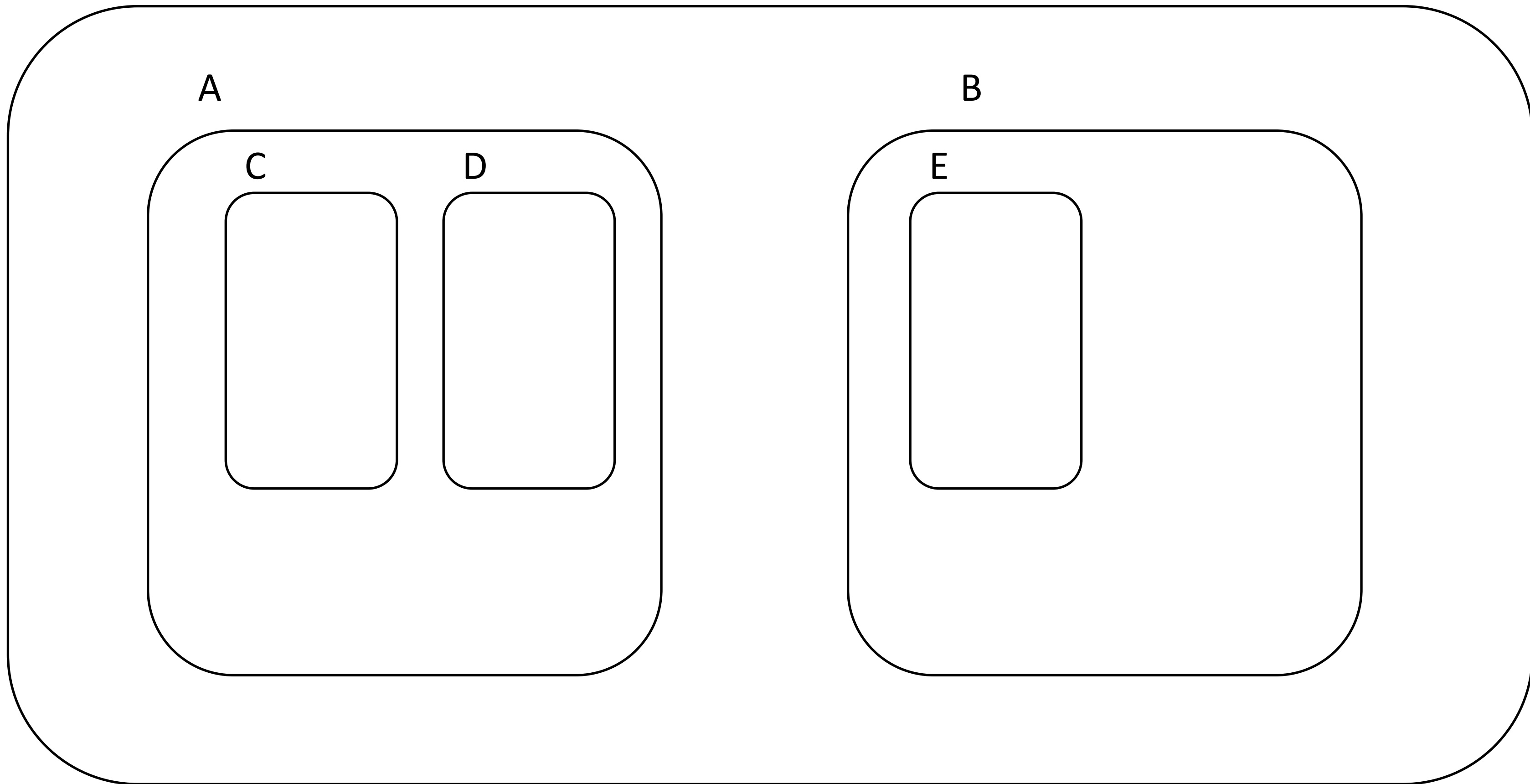
Contour Model

- Language independent model of structure of programs composed of multiple procedures
- Invented by Johnny Johnston in 1971

Definition

- Contour: an executable subprogram with instructions, parameters, local variables, and private working store.
- Name comes from a diagramming method in which subprograms depicted as contours on a topological map.

© 2019, Peter J. Denning



MAIN program declares two sub-programs A and B.
A declares two more sub-programs C and D.
B declares one more sub-program E.

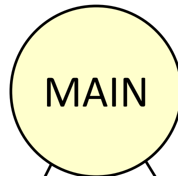
© 2019, Peter J. Denning

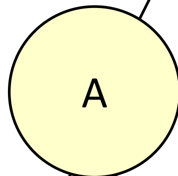
LEVELS

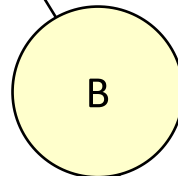
0

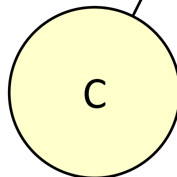
1

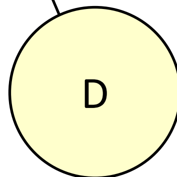
2

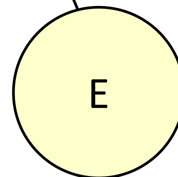












CONTOUR TREE

Contour diagram corresponds to tree of enclosures.

ENVIRONMENT of a contour = path from the node back to the root

e.g.,

E(C) = (C, A, MAIN)
E(A) = (A, MAIN)
E(B) =(B, MAIN)

Block level = distance from root

© 2019, Peter J. Denning

- Contours are a model for block structure in some programming languages
- Contours are a model for type inheritance in object-oriented languages

© 2019, Peter J. Denning

Block Levels

- Block level of a contour or a variable is the level at which the name is declared.
  - Level of MAIN: 0
  - Level of variable x of MAIN: 0
  - Level of A and B: 1
  - Level of variable y of A: 1
  - Level of C, D, and E: 2
  - Higher block level numbers are deeper in the contour tree

© 2019, Peter J. Denning

Execution Sequences

- (C, …) denotes period of invocation of contour C
- "(C" is the moment of call, ")" the return
- Two contour calls are either independent or one embedded in the other (no overlapping)
  - (C1, …) … (C2, …)
  - (C1, … (C2, …) …)
- Activation record exists only during (C, …)

© 2019, Peter J. Denning

Environment of a contour

- E(C) is all the contours mentioned on the path from C to the root of the contours tree
- All elements of E(C) are visible from C and can be accessed from C
  - "Elements of a contour" includes variables and other contours defined in it
- All other elements of the tree are not visible and cannot be accessed

© 2019, Peter J. Denning

- In the example given earlier:
  - MAIN contains variable x; all embedded contours can refer to x.
  - If A also has a variable x, C and D see A's x but not MAIN's x
  - MAIN can refer to its own x, but not A's
  - A contains variable y; C and D can refer to y, but not to B, E, or MAIN
  - MAIN declares A and B; all embedded contours can call A and B.
  - C cannot call E. E cannot call C or D.
  - No contour can see inside any contour at a higher numbered block level

© 2019, Peter J. Denning

Relation with OS Levels

- Restriction that contour can access only the names in its environment similar to the OS level requirement of only downward calls.
- Blocks levels resemble OS levels in that you can access a lower block level from a higher, but not vice versa.
- Blocks levels not exactly the same because contours can only access closest instance of a name; in OS a level can access any lower-level instance.

© 2019, Peter J. Denning

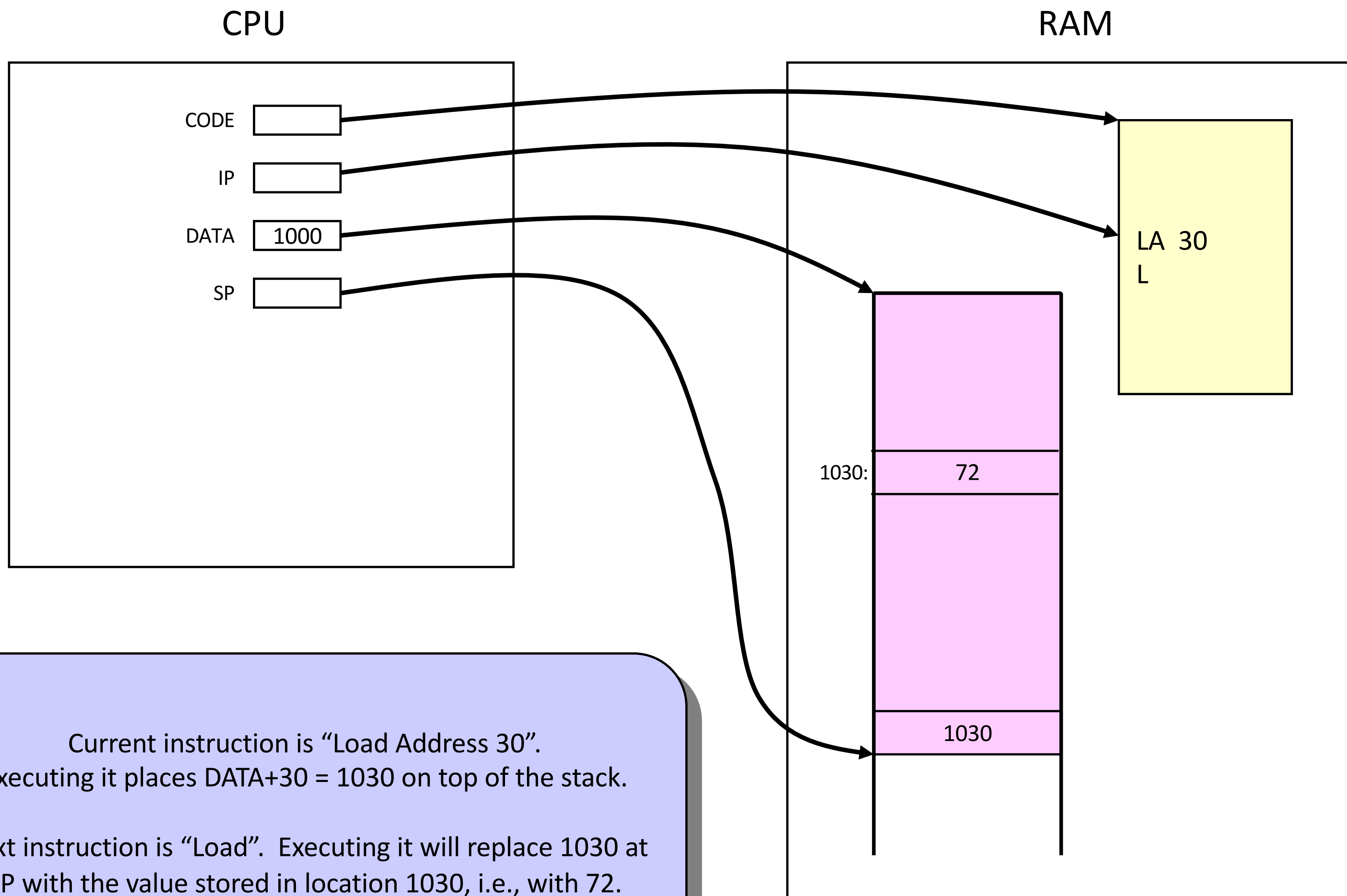
Differences with previous model

- The standard procedure call model, Ch 11.1, restricts any procedure to its local variables – all contained in its activation record, AR.
- The contour model allows procedures to access names in ARs of enclosing contours.
- To do so, maintain DISPLAY, a set of registers that point to the bases of all ARs whose contours enclose the current AR.

© 2019, Peter J. Denning

Implementation

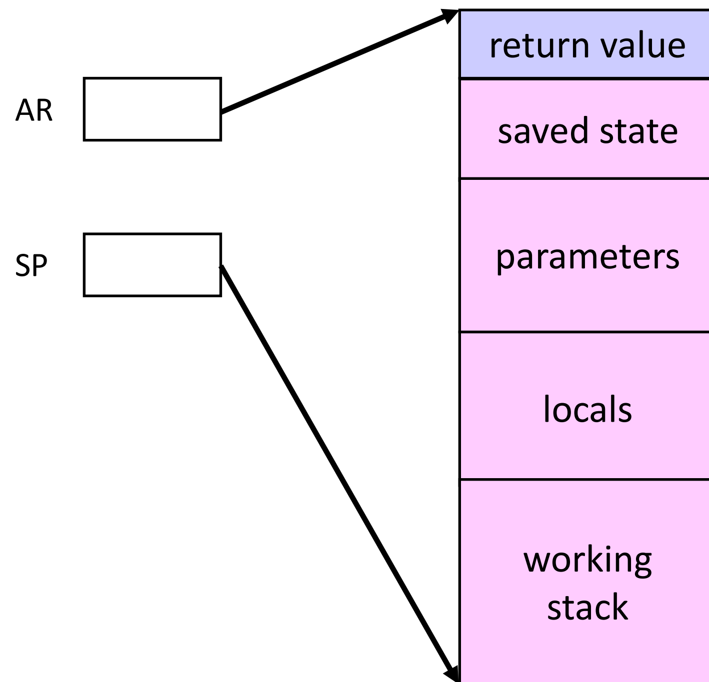
- Diagrams on the next pages show examples of the configurations of stacks, activation records, and contours



Current instruction is “Load Address 30”.
Executing it places DATA+30 = 1030 on top of the stack.

Next instruction is “Load”. Executing it will replace 1030 at SP with the value stored in location 1030, i.e., with 72.

© 2019, Peter J. Denning



STRUCTURE OF AR IN STANDARD MODEL

Starts with slot reserved for return value

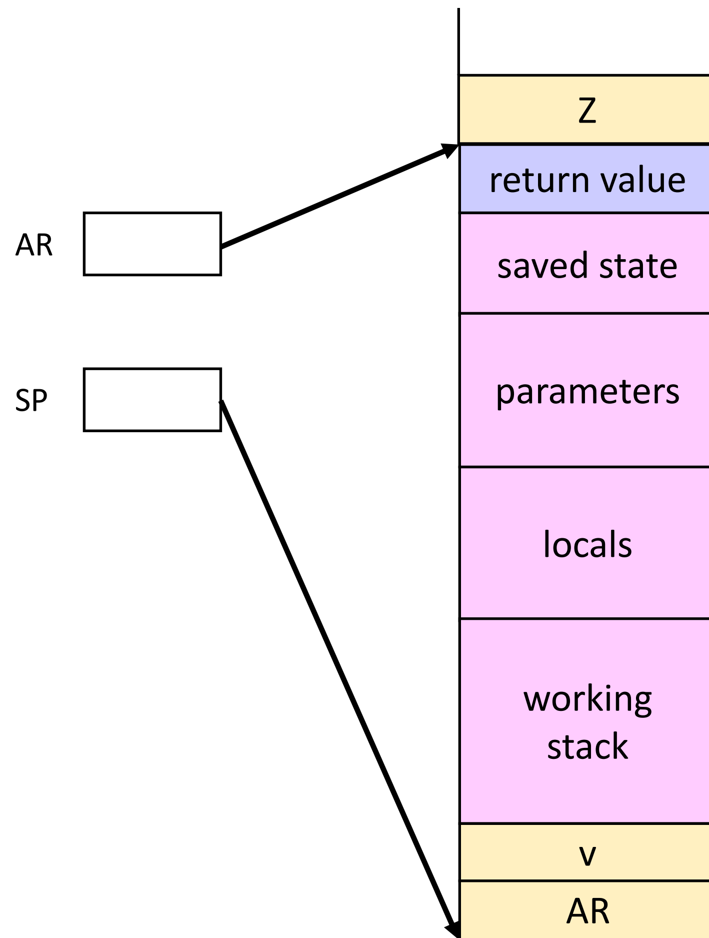
Saved state area contains return state to restore caller's environment (saved IP and AR)

Parameter area contains parameters of the call.

Locals area contains local variables of the procedure.

Working stack contains the temporary store, managed as stack. NOTE: working stack is not part of the AR template. The template is a header of the full AR of the procedure.

© 2019, Peter J. Denning



RETURNING A VALUE

Frame element AR is the target for the return value v. Address of variable Z to hold the return value is placed by “LA Z” just prior to call. After the return the stack will have configuration S Z v and a ST instruction will place the value in the local variable Z

When the called procedure is ready for return it leaves the return value on top of the stack. With these instructions, it moves that value to the return-value slot and returns:

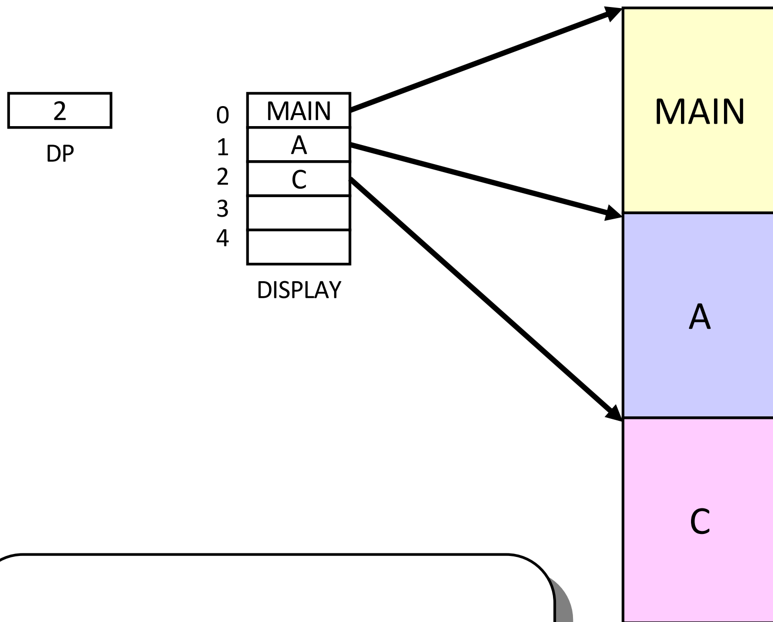
L AR	contents of AR to top stack
EXC	exchange the top two elements
ST	leave v in the return-value slot
RET	return to caller

© 2019, Peter J. Denning

(MAIN, … (A, … (C, .. | (B, …) …) …) …)

t1

t1



The registers of the DISPLAY and the display pointer DP replace the single register DATA shown previously.

DISPLAY contains the current environment (at time t1). Environment E(C) = (C,A,MAIN) is represented as a list of the bases of the current and enclosing ARs.

DP contains the current block level in static tree.

DISPLAY[DP] is base of current AR.

Extend the definition of LA to include the block level k of the contour in which an address x is to be interpreted:

LA k,x

This instruction puts DISPLAY[k]+x on stack. Note that k≤DP because no contour can see variables deeper in the tree. To access a local variable of the current AR:

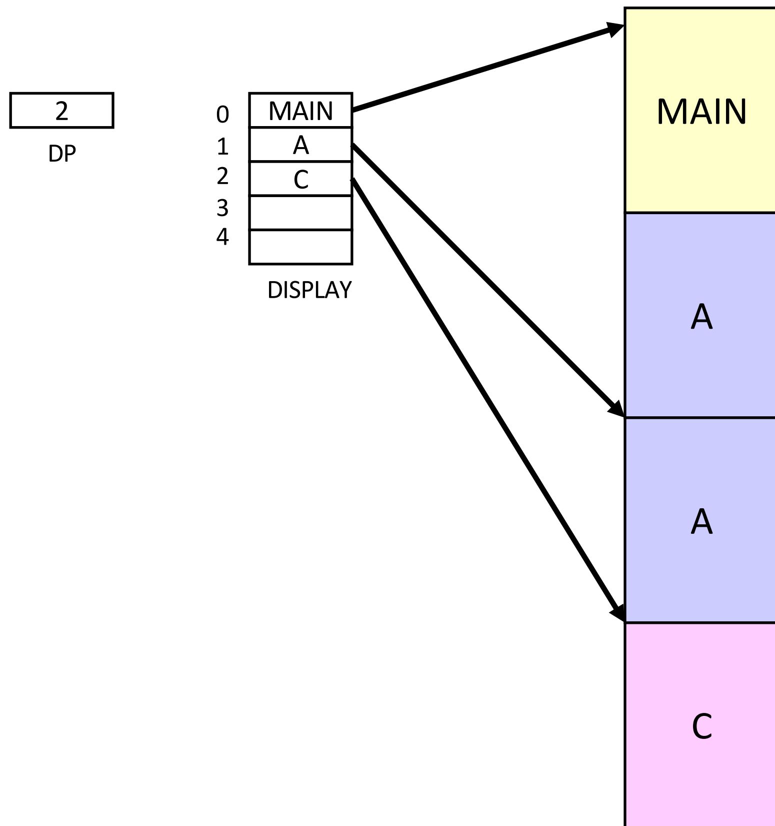
LA DP,x

© 2019, Peter J. Denning

(MAIN, … (A, … (A, … (C, … (B, …) …) …) …) …)

t1

t1



More frames may be on the stack than are indicated by the display. In this case, a recursive call on A leaves two versions of A's AR.

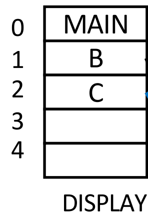
Diagram illustrates an extra frame inserted by a recursive call on A. The A pointer in DISPLAY[1] is to the most recent invocation of A.

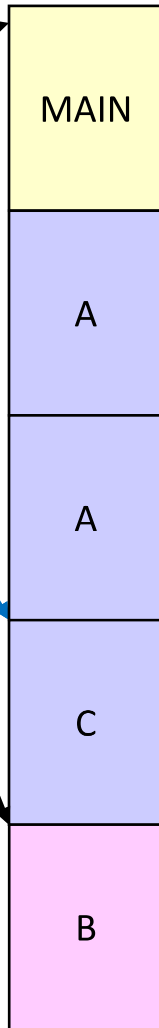
The DISPLAY permits referencing statically-enclosing contours no matter what the dynamics of contour calls have been.

(MAIN, … (A, … (A, … (C, … (B, …) …) …) …) …)

t2

t2





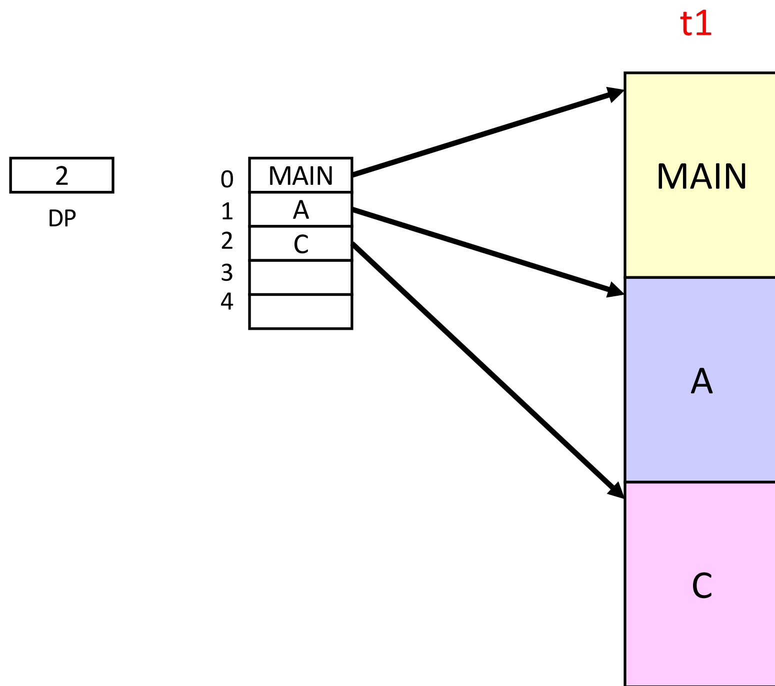
Now C has called B, which is visible because it is declared in MAIN. The environment E(B)=(B,MAIN) is shown in the DISPLAY.

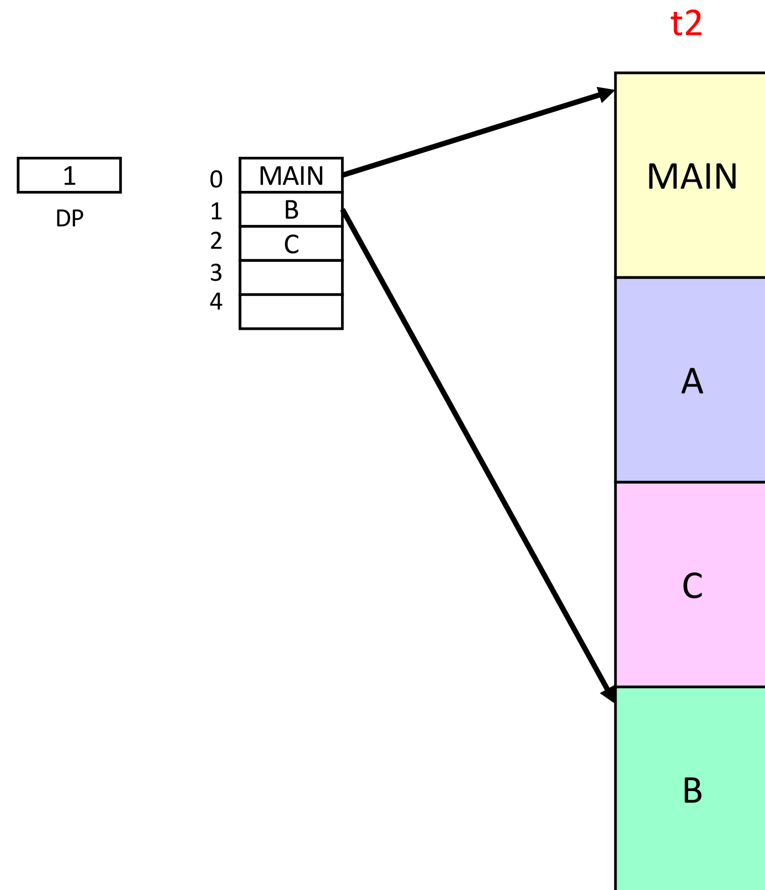
B can refer to its own variables or MAIN, but not to variables in A or C. The pointer to C remains in the DISPLAY but cannot be used because of the restriction that contours cannot reference deeper levels.

© 2019, Peter J. Denning

(MAIN, … (A, … (C, … (B, …) …) …) …)

t1 t2



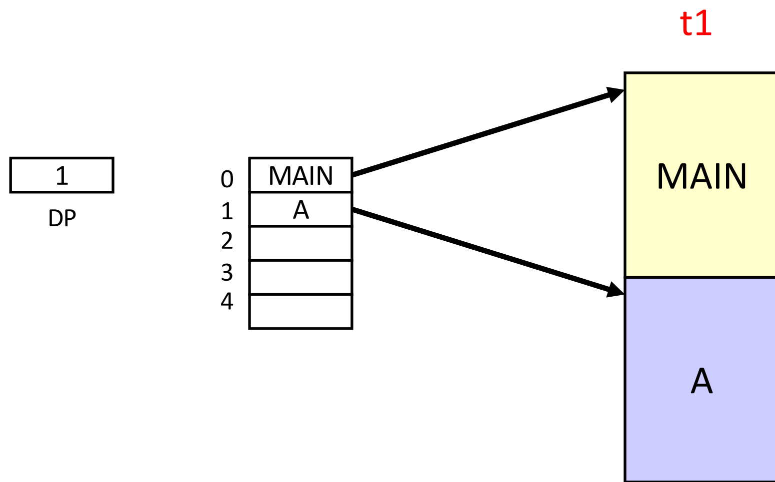


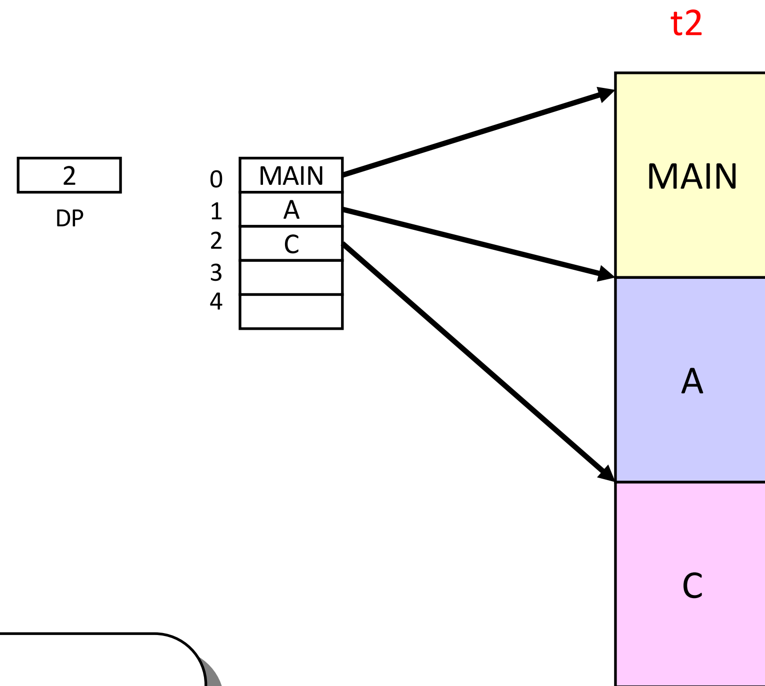
This example shows the stack just before and after the call on B. Again, note that C remains in the DISPLAY but is not accessible .

© 2019, Peter J. Denning

(MAIN, … (A, … (C, … (B, …) …) …) …)

t1 t2

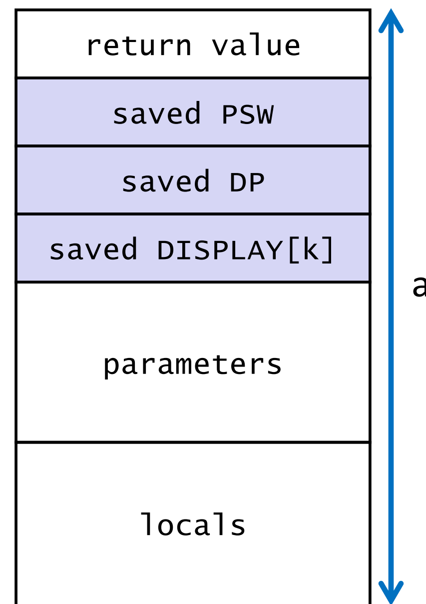




This example shows the stack just before and after the call on C. A can call C because C is defined within A. After the call, the display pointer has increased by 1 and the display has added one more pointer.

© 2019, Peter J. Denning

AR template structure



Prior to the call, the stack configuration is S e a k, where e = entry of called procedure, a = length of AR template, and k is the block level of the called procedure.

The template for the AR header is almost the same as in the standard model, but the "saved AR" has been replaced by the pair (saved DP, saved DISPLAY[k])

PSW (program status word) contains IP, kernel mode, and interrupt masks

© 2019, Peter J. Denning

Operation of CALL and RET

CALL =

```
K = pop Mem[SP]
B = SP-(pop Mem[SP])
Mem[B+1] = IP
Mem[B+2] = DP
Mem[B+3] = DISPLAY[K]
IP = pop Mem[SP]
DP = K
DISPLAY[K] = B
```

For A the AR template, changes stack from S e a k to S A and sets IP=e

Prior to call, DISPLAY[DP] points to base of caller's AR; after, DISPLAY[k] points to base of called AR and DP=k

Saves the caller IP, DP, and DISPLAY[k] in slots 1, 2, and 3 of the new AR

RET =

```
SP = DISPLAY[DP]
DISPLAY[DP] = Mem[SP+3]
DP = Mem[SP+2]
IP = Mem[SP+1]
```

Restores the caller state by reloading the IP, DP, and DISPLAY[k] registers

Leaves SP at base of called AR, which holds the return value

EXAMPLE CALLING SEQUENCE
(simple function call)

Let uppercase denote a variable and lowercase its location offset in its AR. Let X, Y be local variables of a procedure at block level j. Function F is defined at level k ≤ j. Function invocation Y = F(X) is compiled as shown, constructing the new frame on top of the stack.

```
LA  by,y            address of Y on stack (Y at block level by≤j)
SP = SP+4           reserve for return value, caller state
L   DP,x            load parameter X value (X at current block level DP)
L   locals          one or more L to load local values
LA  f               load entry point of procedure F (in CODE segment)
LA  a               load a = size of AR template
LA  k               load k = block level of F
CALL                invoke procedure
ST                  store result in Y (stack configuration S y v)
```

© 2019, Peter J. Denning

EXAMPLE CALLING SEQUENCE
(function call with arithmetic expression as parameter)

Let uppercase denote a variable and lowercase its location in its AR. Let X, Y, Z be local variables of a procedure at block level j. Function invocation Y = F(X+Z) is compiled as shown, constructing the new frame on top of the stack. Thus, we can have an arithmetic expression in place of a parameter.

```
LA  by,y            address of Y on stack (Y at block level by≤j)
SP = SP+4           reserve for return value, caller state
L   DP,x            load X value (X at block level j=DP)
L   DP,z            load Z value (Z at block level j=DP)
A                   X+Z value now on stack in parameter slot
L   locals          one or more L to load local values
LA  f               entry point of procedure F (in CODE segment)
LA  a               load a = size of AR template
LA  k               load k = block level of F
CALL                invoke procedure
ST                  store result in Y (stack configuration S y v)
```

EXAMPLE CALLING SEQUENCE
(function call with another function call as parameter)

Let uppercase denote a variable and lowercase its location in its AR. Let X, F, and G be local names at block level j=DP. Function invocation Y = F(G(X)) is compiled as shown, constructing the new frame on top of the stack. Thus, we can include a function call to obtain value of a parameter: the blue code "interrupts" the AR setup of the black code to compute G(X) in the parameter 1 position of F's AR.

```
LA   by,y          address of Y on stack (Y at block level by≤j)
SP = SP+4          reserve for saved state of caller of F
SP = SP+4          reserve for saved state of caller of G (i.e., F)
L    DP,x          load x value
L    locals        one or more L for locals of g
L    g             load entry point of g (in CODE segment)
L    ag            load size of AR for g
L    bg            load block level at which g defined (bg ≤ j+1)
CALL               call G(X), leaving value in parameter 1 slot of F
L    locals        one or more L to load local values of f
LA   f             load entry point of f (in CODE segment)
LA   af            load size of AR for f
LA   kf            load block level at which f defined (bf ≤ j)
CALL               invoke procedure, leaving v=F(G(X)) on stack
ST                 store result in Y (stack configuration S y v)
```

© 2019, Peter J. Denning

Summary

- Introduced contour model for block structured languages that allow non-local references to enclosing blocks.
- Noted a similarity between visible lower OS levels and visible lower block levels
- Introduced DISPLAY and display pointer DP to keep track of bases of ARs of all blocks enclosing the current block
- Modified the LA instruction to refer to offsets in enclosing blocks
- Modified the CALL to save DISPLAY[k] and DP rather than just AR
- Modified the RET to restore caller's DISPLAY[k] and DP
- As before RET leaves computed function value on top of stack, from where it can be saved to a local variable after return

© 2019, Peter J. Denning