Programming Languages Recitation

Activation Records, Links, Call stack and Closures

Arpit Jain

Computer Science Department
Courant Institute of Mathematical Sciences
New York University

arpit.jain@cs.nyu.edu

September 25, 2014

Overview

1. Introduction
2. Activation Record
3. Links
   - Dynamic Links
   - Static Links
4. Closures
5. Parameter Passing

Storage classes

- Static
- Heap
- **Stack**

Storage Classes

Static

- Global constants
- Compiler generated data
- Variables remain bound to the memory cell throughout the execution
- $\oplus$ Efficiency
- $\ominus$ Lack of flexibility

Storage Classes

Heap

- Data that may outlive the call to the procedure
- ⊕ No limit on memory size
- ⊖ No guaranteed efficient use of space (Fragmentation)
- ⊖ Relatively Slower access
- ⊖ No automatic memory management

Storage Classes

Stack

- Names local to a procedure
- ⊕ Very fast access
- ⊕ Don't have to explicitly deallocate variables
- ⊕ Space is managed by CPU
- ⊖ Limit on stack size (OS dependent)
- ⊖ Local variables only

Run time organization

- Formal - parameters of the function. foo(int a, int b)
- Actual - arguments of the function. foo(i, j)

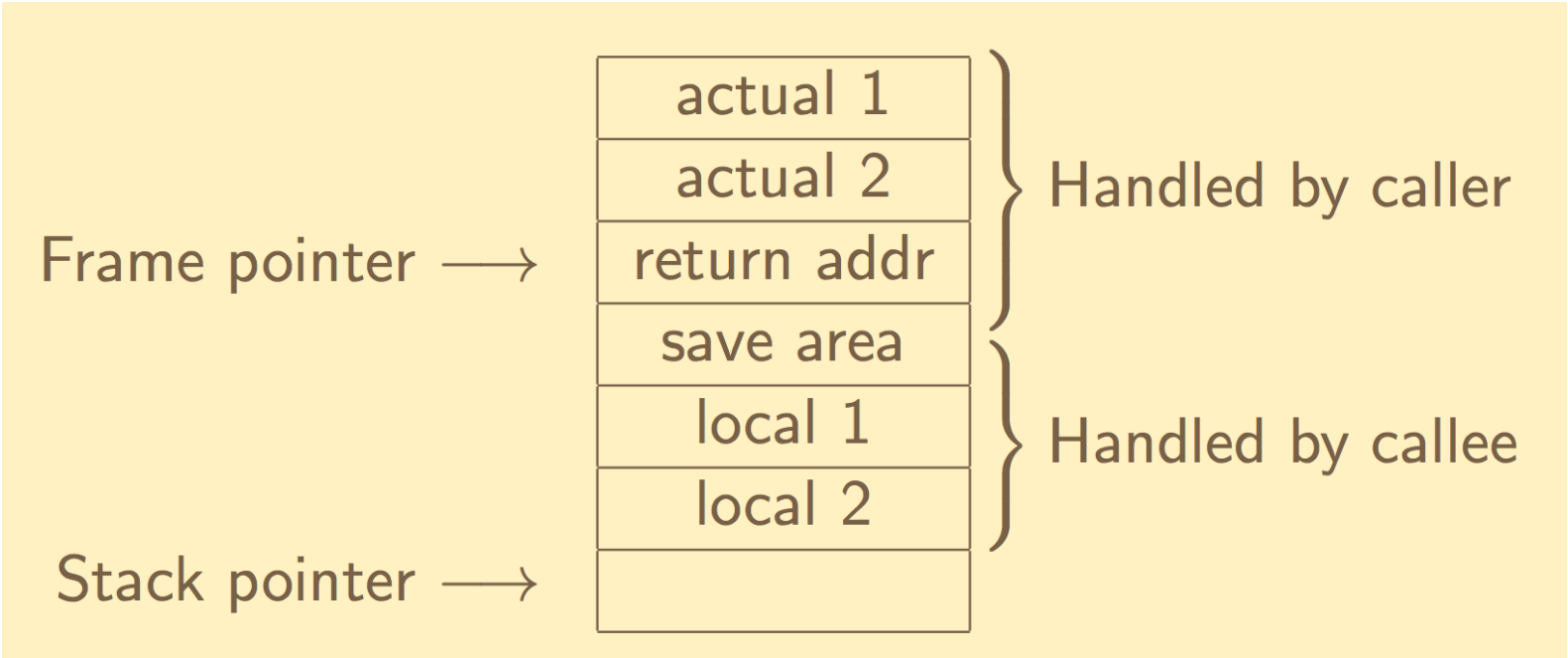
1. Each subprogram invocation creates an activation record
2. Caller: place actuals on stack, return address, linkage information, then transfer control to callee
3. Prologue: save registers, allocate space for locals
4. Epilogue: place return value in register or stack position, update actuals, restore registers, then transfer control to caller

Activation Record Layout

- Stack Pointer - top of the stack. All addresses smaller than this are garbage and greater and equal are valid
- Frame Pointer - value of stack pointer just before the function was called.

But why do we need a frame pointer ?

Activation Record Layout



[1]

[1] Courtesy Prof. Plock

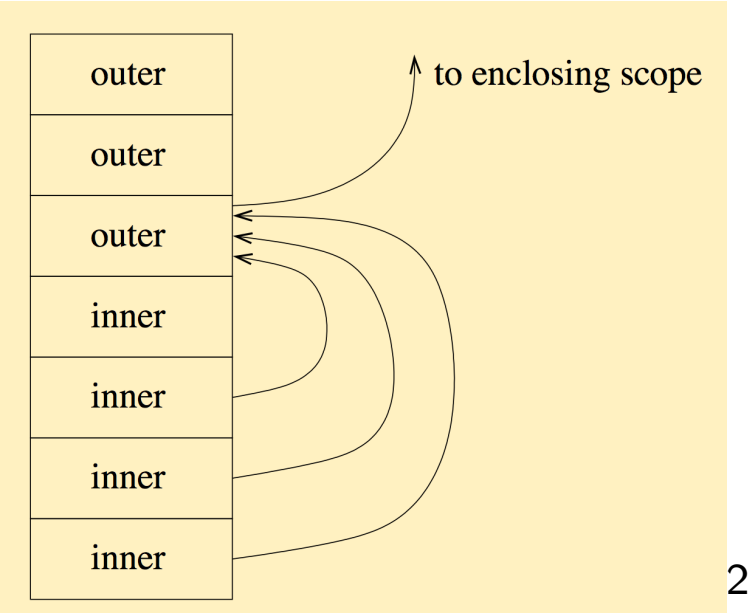
Dynamic Links

- Dynamic link points to the top of the caller
- The saved value of the frame pointer which will be restored on subroutine return, is called the dynamic link

Static Links

- We need to be able to access the variables stored in previous activation records in the stack.
- Pointer to activation record of statically enclosing scope
- To retrieve entity n scopes out, need n dereference operations.

Alternative : Displays



[2]

[2] Courtesy Prof. Plock

First-class function

First-class function

- Passing functions as arguments to other functions
- Returning them as the values from other functions
- Assigning them to variables or storing them in data structures

```
type Ptr is access function (X: Integer) return
 Integer;
function Make_Incr (X: Integer) return Ptr is
 function Incr (Base: Integer) return Integer is
  begin
     return Base + X;
  end;
 begin
 return Incr'access;
 end;
Add_Five: Ptr := Make_Incr(5);
Total: Integer := Add_Five(10);
```

Closures

- Closure is a tuple of (pointer to function, environment of the function)
- Put it on Heap

Parameter Passing

- By value: formal is bound to value of actual
- By reference: formal is bound to location of actual
- By copy-return: formal is bound to value of actual; upon return from routine, actual gets copy of formal
- By name: formal is bound to expression for actual; expression evaluated whenever needed; writes to parameter are allowed (and can affect other parameters!)
- By need: formal is bound to expression for actual; expression evaluated the first time its value is needed; cannot write to parameters

Copy vs Reference

```
program example;
var
global: integer := 10;
another: integer := 2;
procedure confuse (var first, second: integer);
 begin
 first := first + global;
second := first * global;
 end;
 begin
confuse(global, another);
```

You can try calling the function confuse both by copy return and by reference. If you call it by reference, after the call: global = 20 another= 400 However if you call by copy return, after the call: global=20 another= 200.