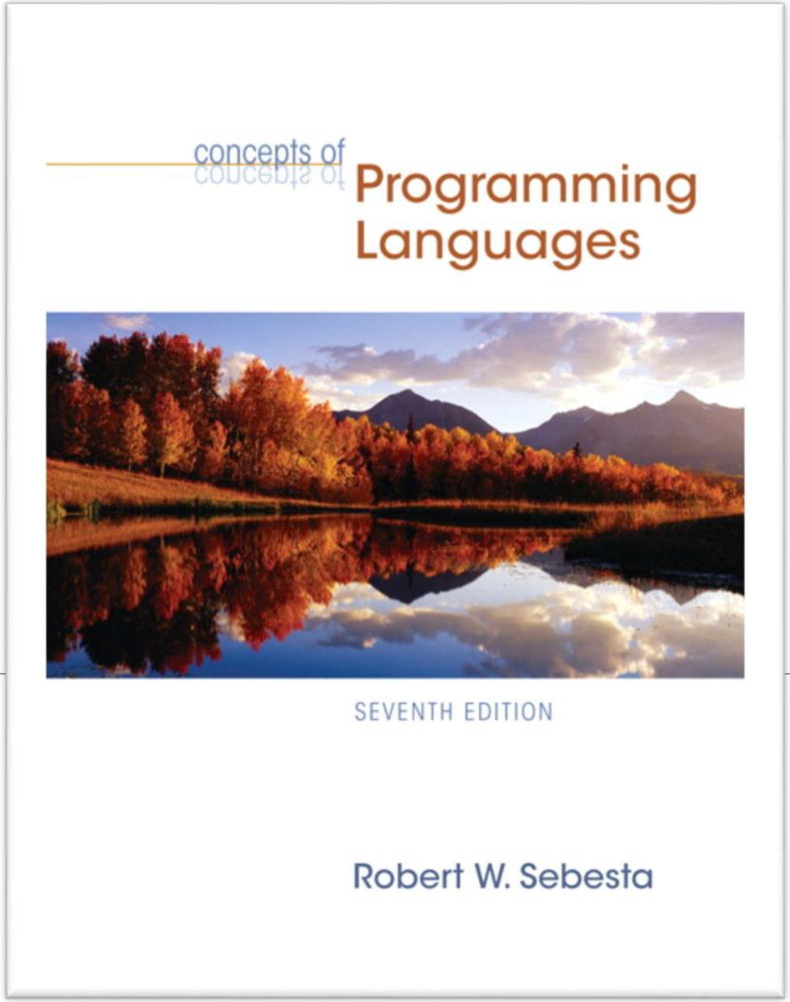
Lecture 5 (Chapter 10)

COP3402 FALL 2015 – 9/14/2015

FROM ADDISON-WESLEY, 2006



Chapter 10 Topics

- The General Semantics of Calls and Returns
- Implementing “Simple” Subprograms
- Implementing Subprograms with Stack-Dynamic Local Variables
- Nested Subprograms
- Blocks
- Implementing Dynamic Scoping

The General Semantics of Calls and Returns

- The subprogram call and return operations of a language are together called its *subprogram linkage*
- A subprogram call has numerous actions associated with it
  - Parameter passing methods
  - Static local variables
  - Execution status of calling program
  - Transfer of control
  - Subprogram nesting

Implementing “Simple” Subprograms: Call Semantics

- Save the execution status of the caller
- Carry out the parameter-passing process
- Pass the return address to the callee
- Transfer control to the callee

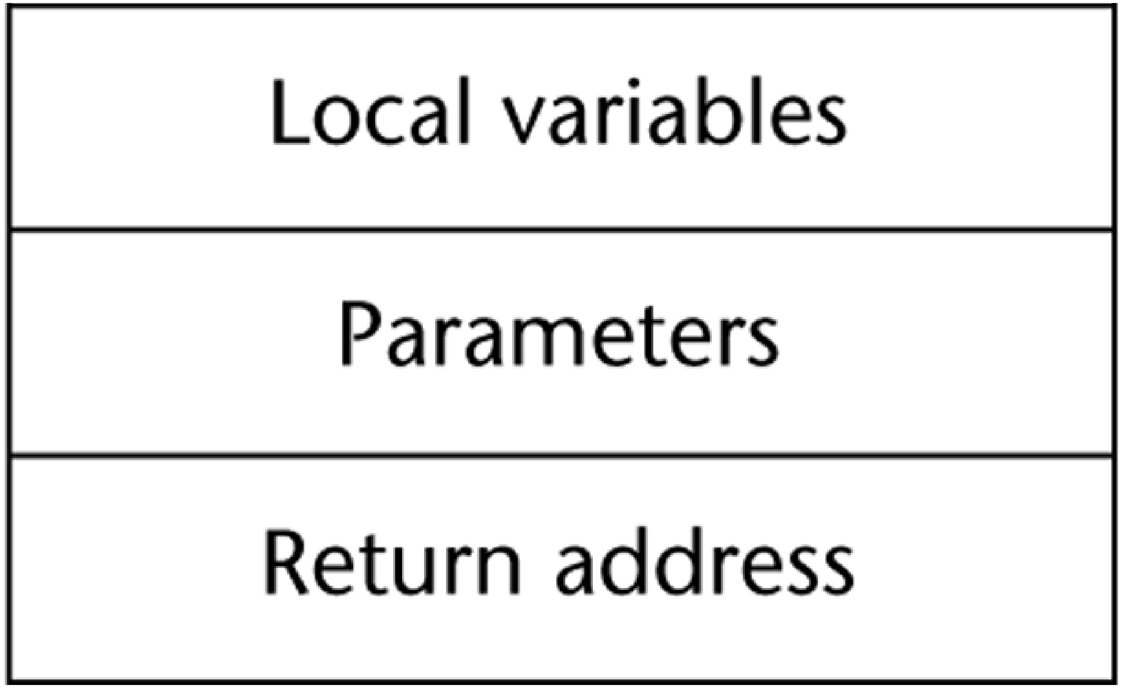
Implementing “Simple” Subprograms: Return Semantics

- If pass-by-value-result parameters are used, move the current values of those parameters to their corresponding actual parameters
- If it is a function, move the functional value to a place the caller can get it
- Restore the execution status of the caller
- Transfer control back to the caller

Implementing “Simple” Subprograms: Parts

- Two separate parts: the actual code and the noncode part (local variables and data that can change)
- The format, or layout, of the noncode part of an executing subprogram is called an *activation record*
- An *activation record instance* is a concrete example of an activation record (the collection of data for a particular subprogram activation)

An Activation Record for “Simple” Subprograms

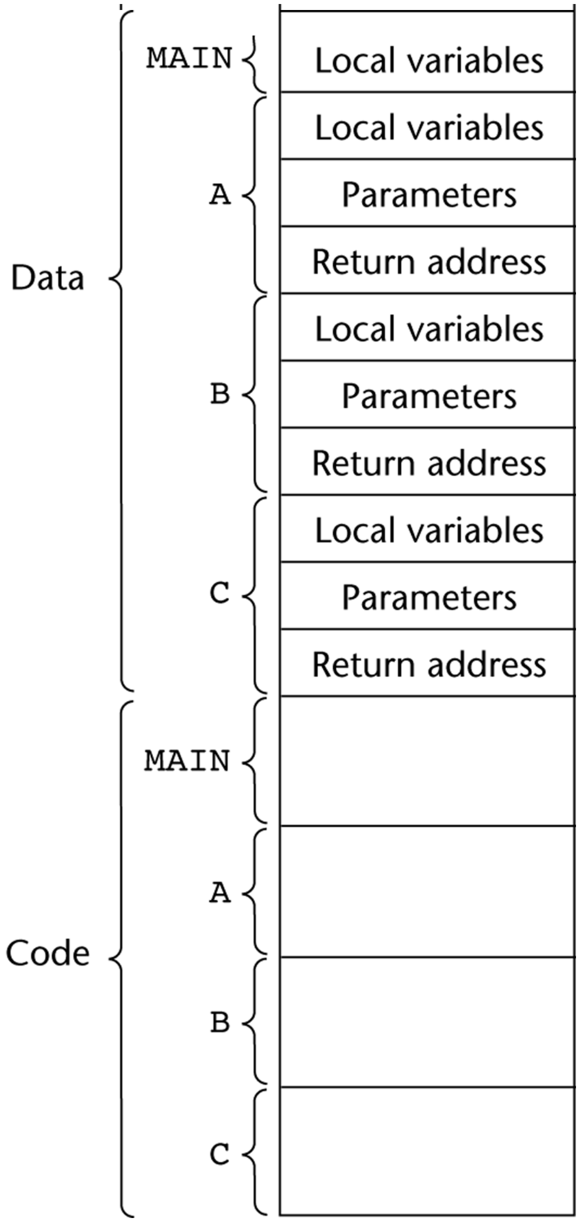


Local variables

Parameters

Return address

Code and Activation Records of a Program with “Simple” Subprograms

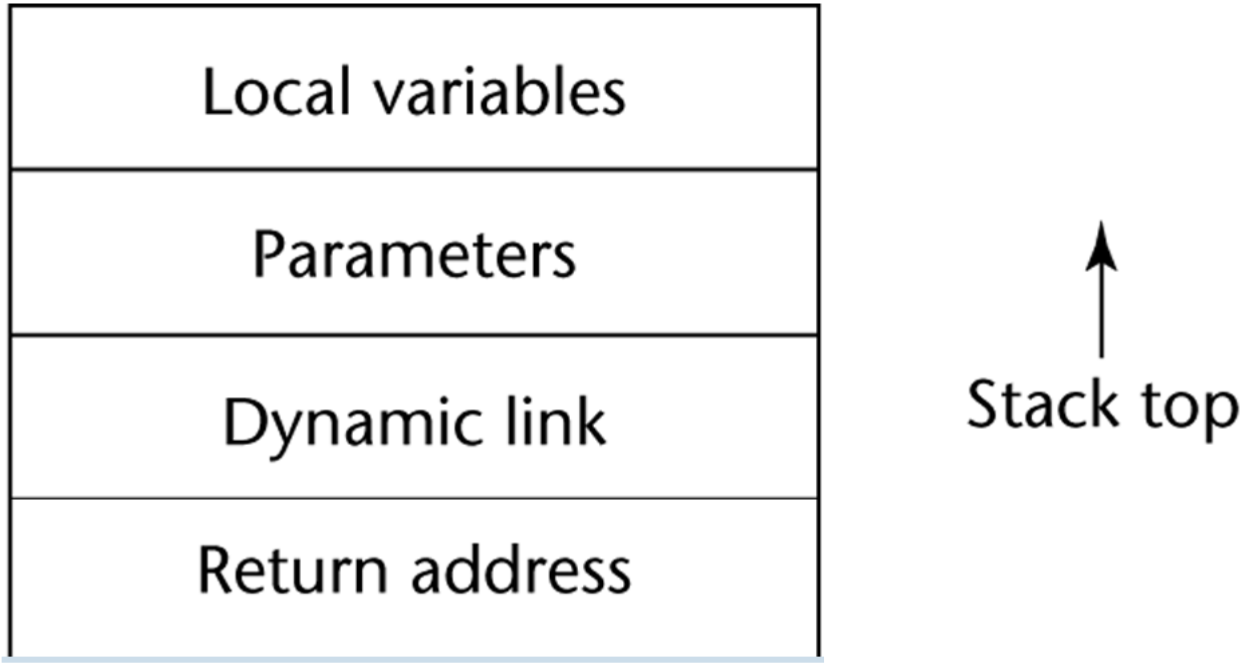


Implementing Subprograms with Stack-Dynamic Local Variables

More complex activation record

- The compiler must generate code to cause implicit allocation and de-allocation of local variables
- Recursion must be supported (adds the possibility of multiple simultaneous activations of a subprogram)

Typical Activation Record for a Language with Stack-Dynamic Local Variables



Local variables

Parameters

Dynamic link

Return address

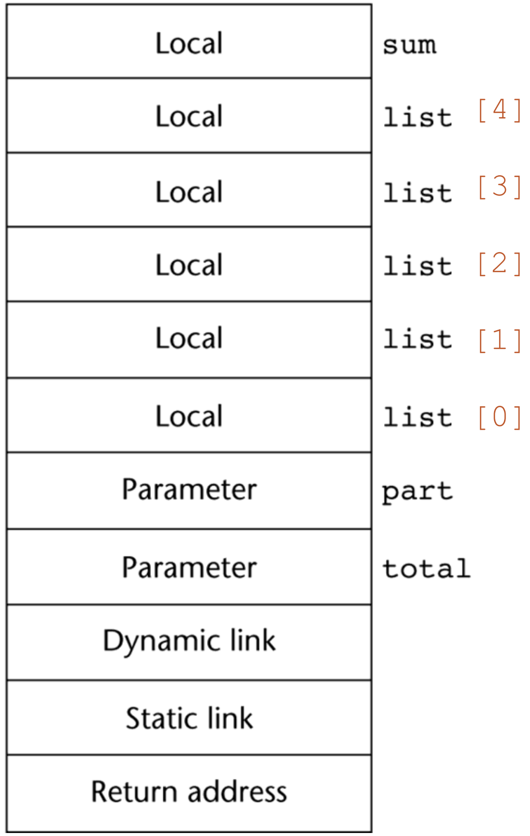
Stack top

Implementing Subprograms with Stack-Dynamic Local Variables: Activation Record

- The activation record format is static, but its size may be dynamic
- The dynamic link points to the top of an instance of the activation record of the caller
- An activation record instance is dynamically created when a subprogram is called
- Run-time stack

An Example: C Function

```
void sub(float total, int part)
{
    int list[4];
    float sum;
    …
}
```



An Example Without Recursion

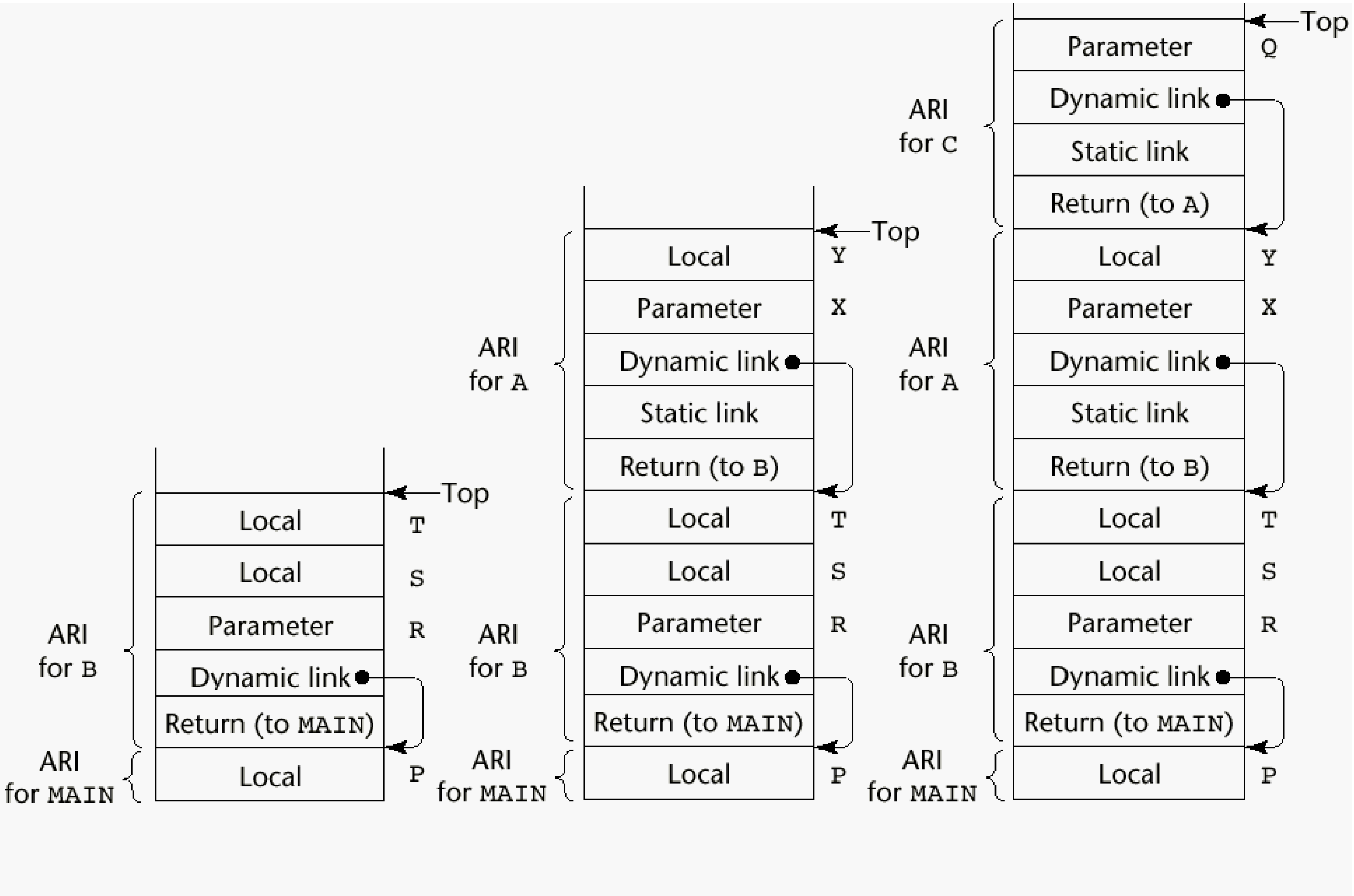
```
void A(int x) {
    int y;
    ...
    C(y);
    ...
}

void B(float r) {
    int s, t;
    ...
    A(s);
    ...
}
void C(int q) {
    ...
}

void main() {
    float p;
    ...
    B(p);
    ...
}
```

main calls B
B calls A
A calls C

An Example Without Recursion



ARI = activation record instance

Dynamic Chain and Local Offset

- The collection of dynamic links in the stack at a given time is called the *dynamic chain*, or *call chain*
- Local variables can be accessed by their offset from the beginning of the activation record. This offset is called the *local_offset*
- The local_offset of a local variable can be determined by the compiler at compile time

Next Time:
Finishing Subprograms