THE GENERAL SEMANTICS OF CALLS AND RETURNS

• The subprogram call and return operations of a language are together called its subprogram linkage

• General semantics of subprogram calls

– Parameter passing methods

– Stack-dynamic allocation of local variables

– Save the execution status of calling program

– Transfer of control and arrange for the return

– If subprogram nesting is supported, access to nonlocal variables must be arranged

The General Semantics of Calls and Returns

• General semantics of subprogram returns:

– In mode and inout mode parameters must have their values returned

– Deallocation of stack-dynamic locals

– Restore the execution status

– Return control to the caller

Call Semantics

- Save the execution status of the caller

- Pass the parameters

- Pass the return address to the callee

- Transfer control to the callee

Return Semantics

– If pass-by-value-result or out mode 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

• Required storage:

– Status information, parameters, return address, return value for functions

Parts

• Two separate parts: the actual code and the noncode part (local variables and data that can change)

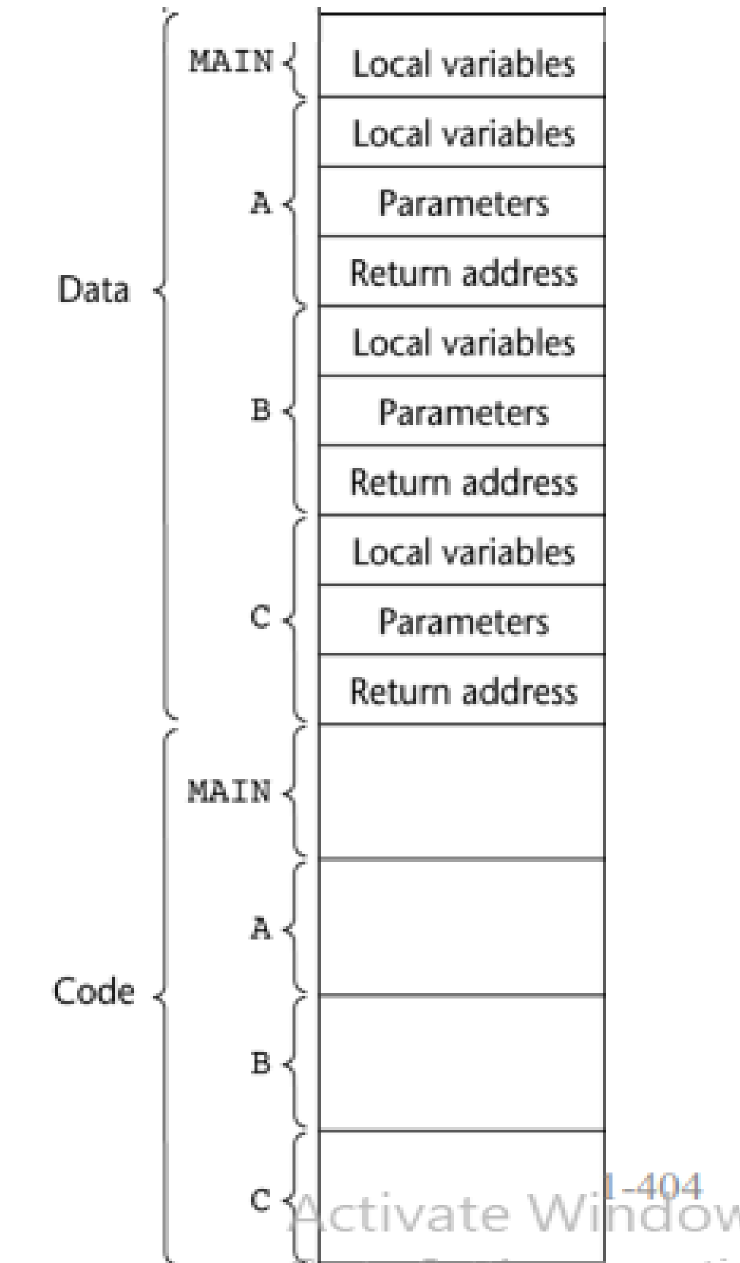
• The format, or layout, of the non-code part of an executing subprogram is called an activation record

• 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



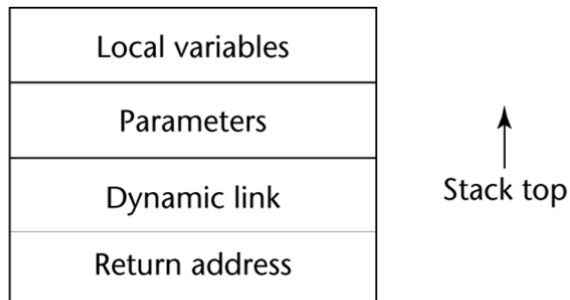
Implementing Subprograms with Stack-Dynamic Local Variables

• More complex activation record

– The compiler must generate code to cause implicit allocation and deallocation 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



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

• Activation record instances reside on the run-time stack

• The Environment Pointer (EP) must be maintained by the runtime system. It always points at the base of the activation record instance of the currently executing program unit

An Example: C Function

```
void sub(float total, int part)
{
int list[5];
 float sum;
…
}
```

Local	sum
Local	list [4]
Local	list [3]
Local	list [2]
Local	list [1]
Local	list [0]
Parameter	part
Parameter	total
Dynamic link	
Return address	
Return address	

Activate Window

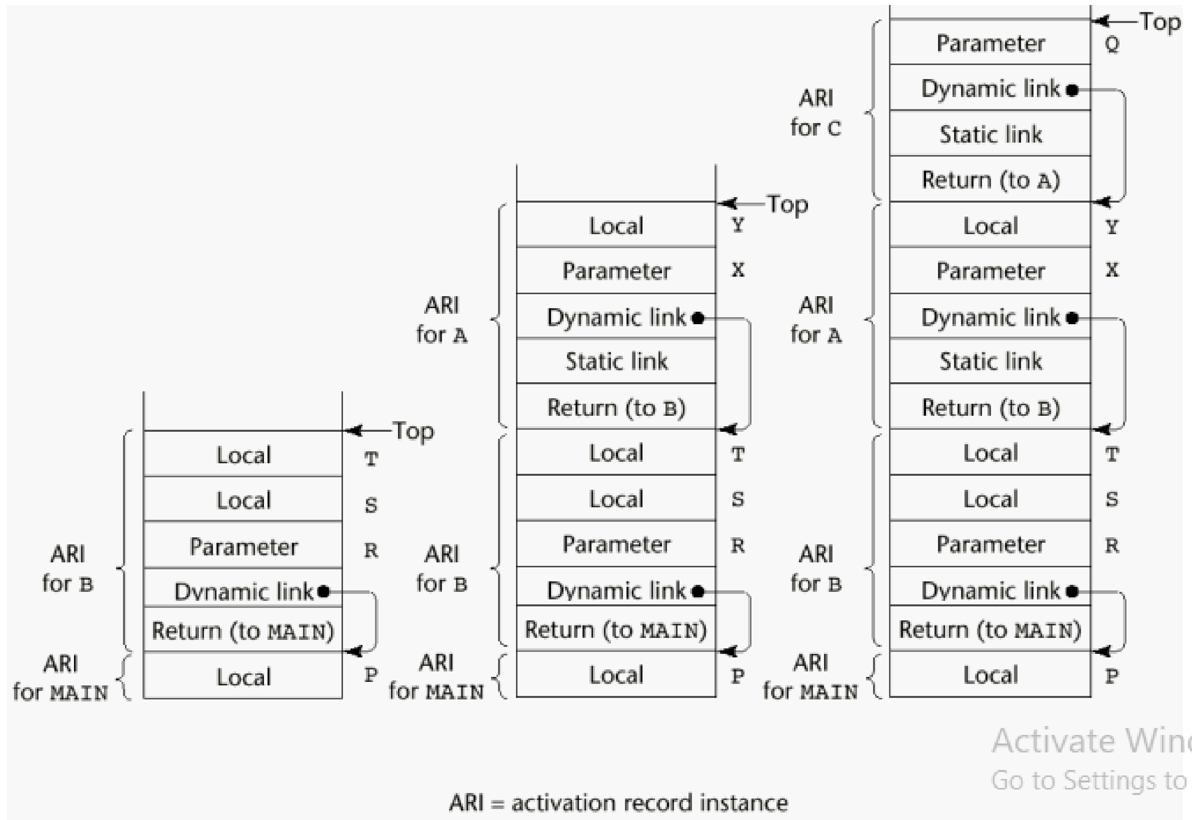
An Example Without Recursion

```
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, whose address is in the EP.This offset is called the local_offset

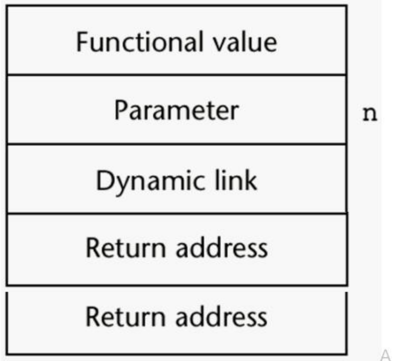
• The local_offset of a local variable can be determined by the compiler at compile time

An Example With Recursion

• The activation record used in the previous example supports recursion, e.g.

```
int factorial (int n) {
 <-----------------------------1
 if (n <= 1) return 1;
 else return (n * factorial(n - 1));
 <-----------------------------2
 }
 void main() {
 int value;
 value = factorial(3);
 <-----------------------------3
 }
```

Activation Record for factorial


Functional value
Parameter
n
Dynamic link
Return address
Return address