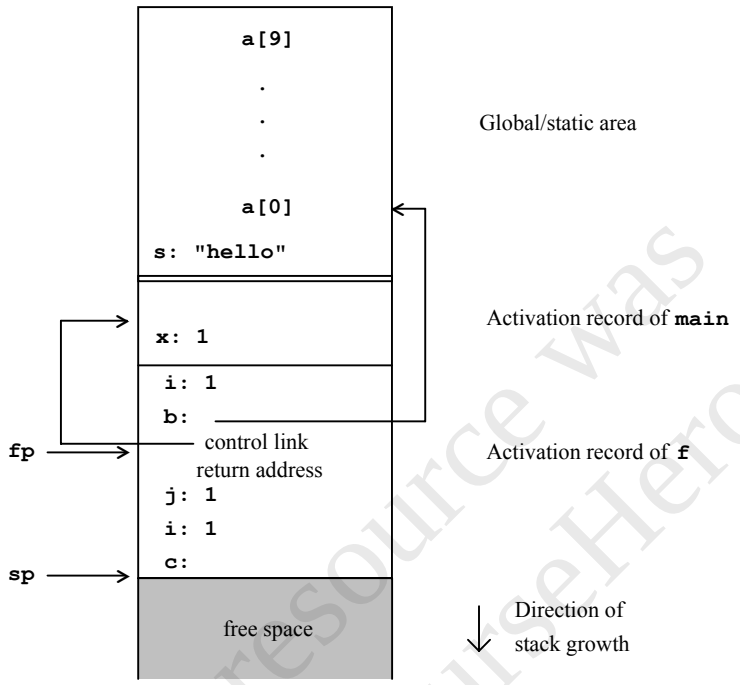
Compiler Construction: Principles and Practice
by Kenneth C. Louden

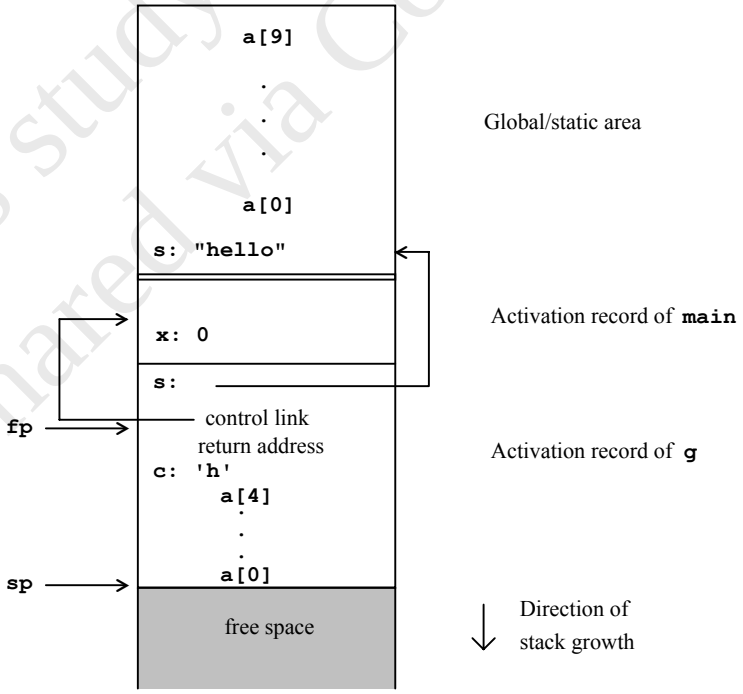
Chapter 7 Exercise Answers

Exercise 7.2

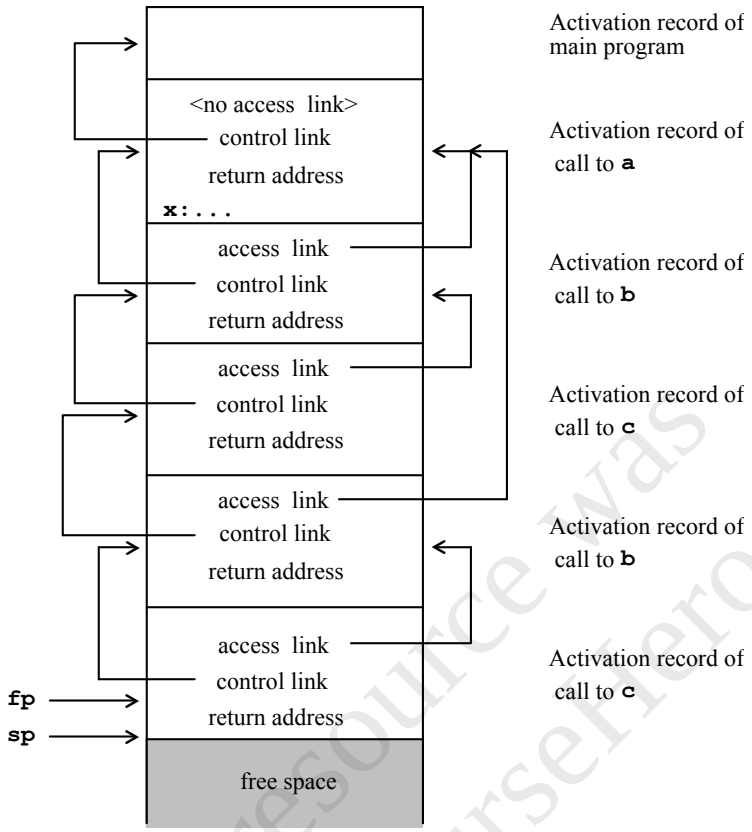
(a)



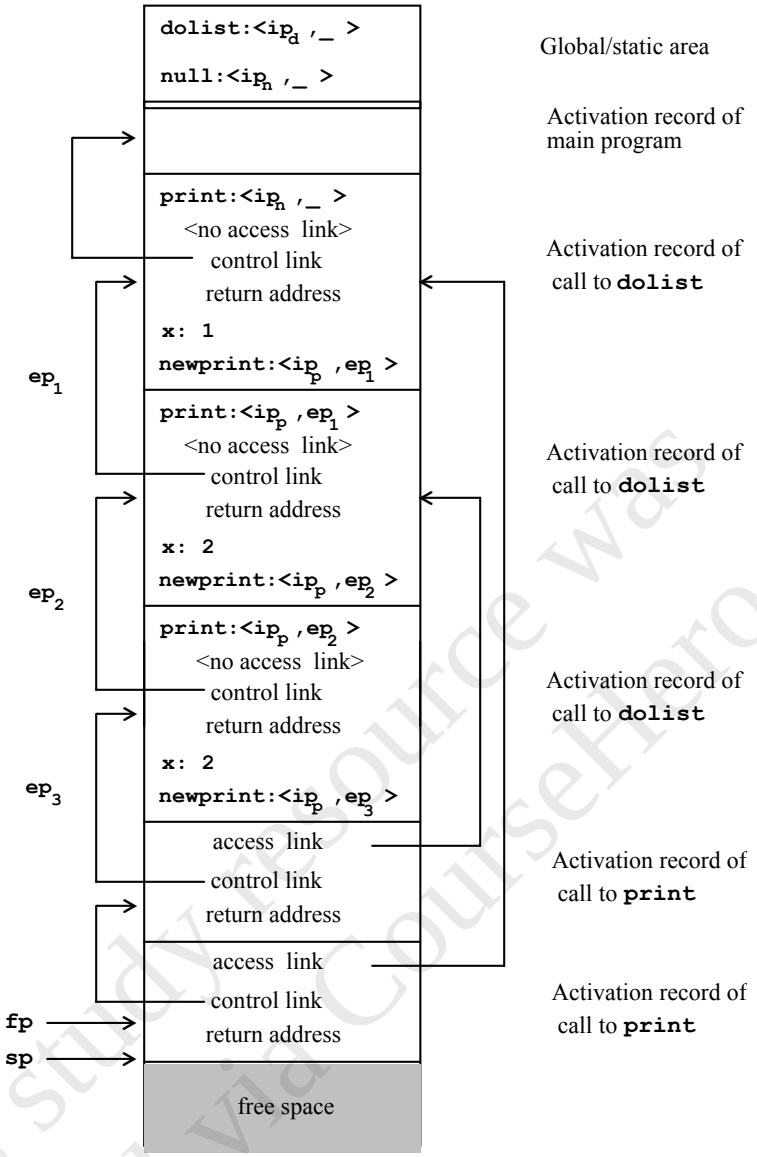
(b)



Exercise 7.4



Exercise 7.6



Exercise 7.8

(a) One way to do this would be to switch the locations of the local variables and the arguments in the activation record. Thus, the local variables would be accessed by positive offset from the fp, and the arguments would be accessed by negative offset, with the first argument pushed onto the stack just below the control link. This would require that the return address be stored at the bottom of the stack (if it is to be pushed using the sp). It also requires that the total size of the local variables of a procedure be known at its call sites (except of course that indirection could still be used for variable-length data). With this organization, the calling sequence would become:

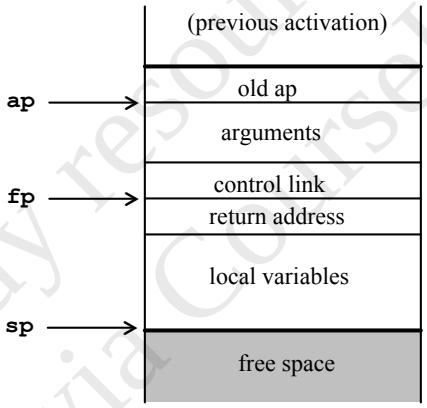
On call:

1. Adjust the sp by subtracting from it the size of the local variables of the procedure.
2. Store the fp in the new activation record.
3. Compute the arguments and push them onto the stack.
4. Store the return address onto the stack.
5. Jump to the code of the procedure to be called.

On exit:

1. Jump to the return address.
2. Pop the arguments.
3. Load the control link into the fp.
4. Add the size of the local variables to the sp.

(b) One way to use an ap pointer would be to have the ap point to the very top of the activation record. Thus, a new ap can be formed by copying the sp just before the arguments are computed. Of course, the old ap must also be stored, and this could be done just before computing the new ap. Then the old ap would be restored using the current ap, rather than the fp. The organization of an activation record then looks as follows:



Now the calling sequence would be as follows:

On call:

1. Push the current ap as the old ap of the new activation record.
2. Move the sp to the ap.
3. Compute and push the arguments.
4. Push the fp as the control link
5. Copy the sp to the fp.
6. Push the return address
7. Jump to the called procedure.

On exit:

1. Copy the fp to the sp.
2. Load the control link into the fp.
3. Jump to the return address.
4. Pop the arguments.
5. Load the old ap into the ap.

Exercise 7.11

(a) The standard C parameter passing conventions are to pass all arguments by value, except that array are considered pointer values, so arrays are actually passed by reference. Thus, the offsets are as follows:

1. `c`: 16
2. `s[7]`: the address of `s` is at offset 12, then 7 is added to this address to find `s[7]`.
3. `y[2]`: –14

(b)

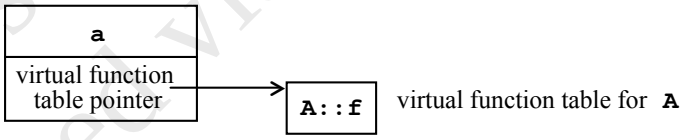
1. `c`: 22
2. `s[7]`: 19
3. `y[2]`: –14

(c)

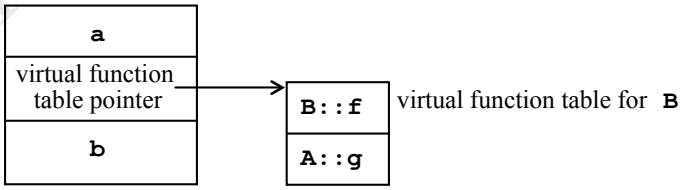
1. `c`: 12
2. `s[7]`: the address of `s` is at offset 8, then 7 is added to this address to find `s[7]`.
3. `y[2]`: –14

Exercise 7.13

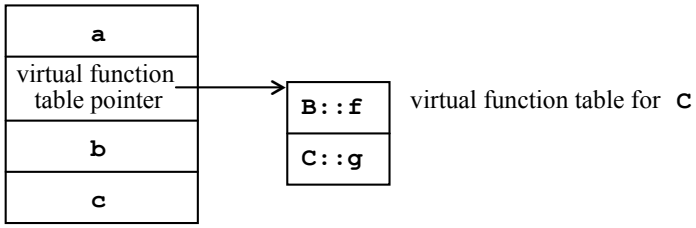
Object of class `A`:



Object of class `B`:



Object of class `C`:



Exercise 7.16

Pass by value: 0 1 2 0

Pass by reference: 1 0 2 0

Pass by value-result: 1 0 2 0 (no recomputation of address of `a[i]` on exit)
1 1 0 0 (with recomputation of address of `a[i]` on exit)

Pass by name: 2 1 –1 0