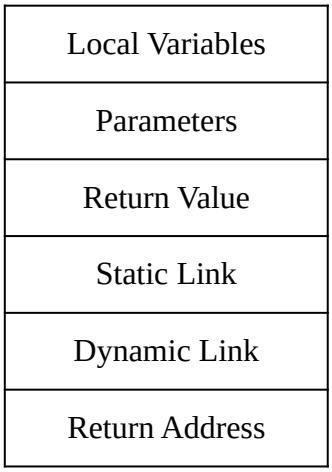
CS 430 Activation

Consider a C-like language that allows nested subprograms, has static scope, and passes parameters by value by default and by reference if the keyword `ref` appears in the declaration of the parameter.

A compiler for this language on a particular architecture uses the following layout for activation records (the stack bottom is at the bottom of the diagram, so it grows upwards):



The system uses an EP (environment pointer) that points at the base of the activation record and a SP (stack pointer) that points at the top of the stack, which is the first open spot not the last used spot.

Use the following conventions for drawing the run-time stack:

- Use brackets and names to indicate the activation records for each active subprogram.
- Mark activation record locations with the name of the variables or parameters stored there, or the following abbreviations:
  - RA = Return Address
  - RV = Return Value
  - SL = Static Link
  - DL = Dynamic Link
- Use source code lines for return addresses.
- Only include return value locations if they are needed in the activation record (that is, if the activation record is for a function).
- Draw static and dynamic links as arrows.
- Record parameter and variable values in the stack.
- Mark the locations of the EP and the SP.

Here is a program in the language described above:

```
01   program P {
02       int x;
03
04       void A() {
05           int a, b, c;
06
07           int F(int x) {
08               int r;
09               b = 5;
10               if (x > 0) r = b + F(x - 1);
11               else       r = b + 1;          // Location 2
12               println(a, b, c);
13               return r;
14           }
15
16           int G(ref int x) {
17               int r;
18               x -= 1;
19               r = F(x) + a;
20               return r;
21           }
22
23           a = x + 1;
24           b = x + x;
25           c = x - 1;         // Location 1
26           a = G(c);
27           println(a, b, c);
28       }
29
30       x = 3;
31       A();
32   }
```

1. Draw the runtime stack when execution of the program above reaches Location 1.

2. Update the runtime stack to show its state when execution of the program reaches Location 2. Don't erase old values or pointers; cross them out and write the new ones beside them.

3. Trace the program through to the end of the execution of the program, crossing out old values or pointers and writing the new ones beside them. DO NOT erase or cross out entire stack frames as the functions return (i.e., assume the data on the stack remains unmodified after it has been deallocated).

4. What is the output of the program?