Taylan Unal (tvu2)

1. (6 points) Consider the following function written in C:

```
int x=3, y=2, z=1;
void foo(int a, int b) {
  x = x + b;
  a = a - b;
}
```

In each of the cases below, write down the values of x, y and z after the following calls to foo(). If necessary, assume that output arguments are copied back to parameters in the left-to-right order.

(a) foo(y,z) where all parameters are passed by value

int x=3, y=2, z=1
foo(int y_a, int z_b):
$x = \overset{3}{x} + \overset{1}{z_b}$ => x=4
$y_a = \overset{2}{y_a} - \overset{1}{z_b}$ => y=1
global variables not updated

Result:
x: 4
y: 2
z: 1

(b) foo(y,z) where all parameters are passed by reference

int x=3, y=2, z=1
foo(int y_a, int z_b):
$x = \overset{3}{x} + \overset{1}{z_b}$ => x=4
$y_a = \overset{2}{y_a} - \overset{1}{z_b}$ => y=1
values of x,y updated

Result:
x: 4
y: 1
z: 1

(c) foo(y,z) where all parameters are passed by value-result

int x=3, y=2, z=1
foo(int y_a, int z_b):
$x = \overset{3}{x} + \overset{1}{z_b}$ => x=4
$y_a = \overset{2}{y_a} - \overset{1}{z_b}$ => y=1
updates only those values that were parameters, so y=1, z same at z=1

Result:
x: 4
y: 1
z: 1

(d) foo(x,y) where all parameters are passed by reference

int x=3, y=2, z=1
foo(int x_a, int y_b):
$x = \overset{3}{x} + \overset{2}{y_b}$ => x=5
$x_a = \overset{5}{x_a} - \overset{2}{y_b}$ => x=3
value of x updated to 3

Result:
x: 3
y: 2
z: 1

(e) foo(x,x) where all parameters are passed by reference

int x=3, y=2, z=1
foo(int x_a, int x_b):
$x = \overset{3}{x} + \overset{3}{x_b}$ => 6
$x_a = \overset{3}{x_a} - \overset{3}{x_b}$ => 0
x is updated to 0

Result:
x: 0
y: 2
z: 1

(f) foo(x,x) where all parameters are passed by value-result

Left to Right

int x=3, y=2, z=1
foo(int x_a, int x_b):
$x = \overset{3}{x} + \overset{3}{x_b}$ => x=6
$x_a = \overset{3}{x_a} - \overset{3}{x_b}$ => x_a=0
only update x

Result:
x: 3
y: 2
z: 1

Stack
Set x to x_a=6
Set x to 0
Set x to x_b=3

2. (2 points) Consider the following C++ program, where &i means i is passed by reference:

```
int bar (int &i) {
  i = i - 2;
  return 2 * i;
}
void foo1 () {
  int x = 3, y = 6, sum;
  sum = x;
  sum = sum + bar(x);
}
void foo2 () {
  int x = 2, y = 7, sum;
  sum = bar(x);
  sum = sum + x;
}
```

&i : pass by reference
i : pass by value

(a) What is the value of sum at the end of the function foo1? Briefly explain why.

foo1()
1) int x=3, y=6, sum; ← initial declaration
2) sum = 3;
3) sum = sum + bar(x)

Assignment Stack

sum	x
~~sum=?~~	~~x=3~~
~~sum=3~~	x=1
sum=5	

bar(&x) x=3, same as outside reference to x
x = x-2 ⇒ x=1. update x to x=1
return 2·x
returns 2

3) sum = 3 + 2 = 5 → sum=5

Since sum is initially set to 3 on line 2, the next line sets sum equal to 3+ the return value from bar(x=3). The value of x is passed by reference, so inside bar, x is updated to 3-2=1. Then, bar returns 2·x where x=1, so returns 2. Therefore, sum is equal to 3+2=5.

(b) What is the value of sum at the end of the function foo2? Briefly explain why.

foo2()
1) int x=2, y=7, sum; ← initial declaration
2) sum = bar(x)

Assignment Stack

sum	x
~~sum=?~~	~~x=2~~
sum=0	x=0

bar(&x) x=2, same as outside reference to x
x = x-2 (2-2) ⇒ x=0. update x to x=1
return 2·1
returns 0

3) sum = sum + x = 0 + 0
sum=0

Since sum is set to the value of bar(x), and bar(x) takes the reference to x, after bar(x=2) runs, it returns 0, setting sum=0. Inside bar, x is also set to 0, and since x is passed by reference, x=0. On line 3, sum is set to sum+x, or 0+0, making the final value of sum=0.

3. (4 points) Consider the Ada program given below.

```
1  procedure Main is
2    A, B, C : Integer;
3    procedure Sub1 (C:Integer) is
4      D, E : Integer;
5    begin -- of Sub1
•6       ...
7    end; -- of Sub1
8    procedure Sub2 (E:Integer) is
9      procedure Sub3 (F:Integer) is
10       C, D: Integer;
11     begin -- of Sub3
•12      Sub1(100);
13     end; -- of Sub3
14   begin -- of Sub2
15     Sub3(E);
16   end; -- of Sub2
17 begin -- of Main
18   Sub2(10);
19 end; -- of Main
```

Ada is a statically scoped language. In the above program, the Main function invokes Sub2; Sub2 invokes Sub3; and Sub3 invokes Sub1. Draw the stack of activation records when the program's execution reaches before line 12 and line 6. For each activation record, include local variables, parameters, the dynamic link, the static link, and the return address.

Dynamic Link points to RA

Static Link points to static Link.

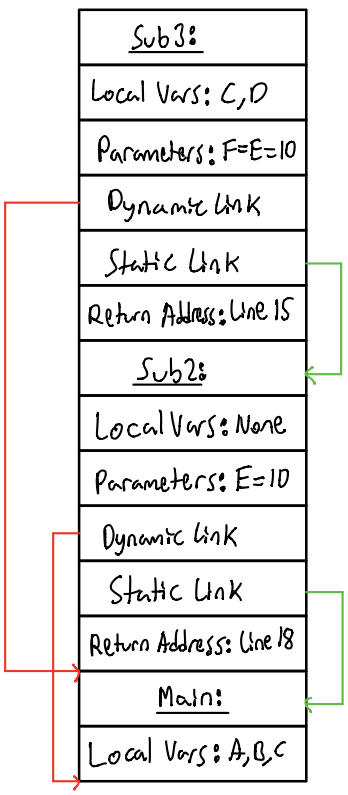
Example Activation Record

Return Value
Local Variables
Parameters
Dynamic Link
Static Link
Source Registers
Return Address

Execution order: Main → Sub2 → Sub3 → Sub1

Before Line 12:

● = dynamic link
● = static link



Before Line 6:

● = dynamic link
● = static link