CSDS 600: Subprogram Written Exercise Answers

Question: Consider the following program written in a C-like language

```
type coord = struct {int x; int y; int z};

function rotate(coord[] a, coord b) {
  if (a[0].x >= a[1].x)
    b.y = a[0].y;
  else
    b.y = a[1].y;

  if (a[0].y > a[1].y)
    b.z = a[0].z;
  else
    b.z = a[1].z;

  if (a[0].z <= a[1].z)
    b.x = a[0].x;
  else
    b.x = a[1].x;
}

void main() {
  coord[] values = { {0, 1, 0}, {1, 0, 1} };

  rotate(values, values[values[0].z];
  print(values);
}
```

Answer: What are the contents of `values` if we use:

A. call-by-value: values stores { `{0, 1, 0}`, `{1, 0, 1}` }**.**
In call-by-value the actual parameters are copied to the formal parameters so the actual parameters do not change.

B. call-by-reference: values stores { `{0, 0, 1}`, `{1, 0, 1}` }
Formal parameter `a` is a reference to `values` while formal parameter `b` is a reference to `values[0]`. Since `values[0].x < values[1].x`, we set `values[0].y` to `values[1].y` so `values` is now { `{0, 0, 0}`, `{1, 0, 1}`}.

Since `values[0].y = values[1].y` we set `values[0].z` to `values[1].z` and so `values` is now { `{0, 0, 1}`, `{1, 0, 1}` }.

Since `values[0].z = values[1].z` we set `values[0].x` to `values[0].x` and there is no change.

C. call-by-value-result: there are two possible answers. Either **values** stores { `{0, 1, 0}`, `{1, 0, 1}` } or **values** stores { `{0, 0, 0}`, `{1, 0, 1}` }.
The actual parameters are copied to the formal parameters so `a = {{0, 1, 0}, {1, 0, 1}}` and `b = {0, 1, 0}}`.

Since `a[0].x < a[1].x` we set `b.y` to `a[1].y`, and so `b` is now `{0, 0, 0}`.

Since `a[0].y > a[1].y` we set `b.z` to `a[0].z`, and so `b` does not change.

Since `a[0].z < a[1].z` we set `b.x` to `a[0].x`, and so `b` does not change.

If the actual parameters are copied back right to left, the copy to `values` overwrites the copy to `values[0]` and so `values` stores { {`0, 1, 0`}, {`1, 0, 1`} }. If the actual parameters are copied back left to right, the copy to `values[0]` overwrites the copy to `values` and so `values` stores { {`0, 0, 0`}, {`1, 0, 1`} }.

D. call-by-name: values will store { {`0, 0, 1`}, {`0, 0, 1`} }.

Since `values[0].x < values[1].x`, we set `values[values[0].z].y` (or `values[0].y`) to `values[1].y`. So now `values` is { {`0, 0, 0`}, {`1, 0, 1`}}.

Since `values[0].y = values[1].y`, we set `values[values[0].z].z` (or `values[0].z`) to `values[1].z`. So now `values` is { {`0, 0, 1`}, {`1, 0, 1`}}.

Since `values[0].z = values[1].z`, we set `values[values[0].z].x` (or `values[1].x`) to `values[0].x`. So, now `values` is { {`0, 0, 1`}, {`0, 0, 1`} }.