Jason Tran

605975912

4/30/23

CS33 HW #4

**Problem 3.70:**

A)

e1.p: **0 bytes**

e1.y: **8 bytes**

e2.x:  **0 bytes**

e2.next: **8 bytes**

B)

The structure requires **16 bytes**.

C)

void proc (union ele \*up) {

up-> **e2.next** = \*(**up -> e2.next -> e1.p**) - (**up -> e2.next -> e1.y**);

}

**Problem 2.89:**

1. (float) x == (float) dx

**Can Yield 0:** Since floating an int could have some problems. If x is something like tmax and you cast it to the float, we know that the bits in the float are sectioned off 3 ways and all the bits aren’t interpreted as the actual number, this means it can’t = to the original tmax without some sort of rounding.

1. dx — dy == (double) (x-y)

**Can Yield 0:** If X represented tmax and y represented tmin then there would be some overflow errors even before it gets casted to a double. This would mean there would be an inequality with the left side and right side.

1. (dx + dy) + dz == dx + (dy + dz)

**Always Yields 1:** Since dx,dy,dz are all integers casted to doubles, which means the original are preserved with spare bits. So adding in these two different orders will yield one with the associative property of addition and the extra bits that you get with it being a double.

1. (dx \* dy) \* dz == dx \* (dy \* dz)

**Can Yield 0:** If dx, dy, and dz were all numbers tmax or very close to tmax, multiplying them even with them being integers casted to doubles will result in overflow as there won’t be enough bits and there will be rounding inequalities. Especially if you multiply in a different order.

1. dx / dx == dz / dz

**Can Yield 0:** If dx is 0 and dy is anything else, dividing by 0 will result in NaN, which won’t equal to the other side, 1.