# CS370 – ASSIGNMENT #6

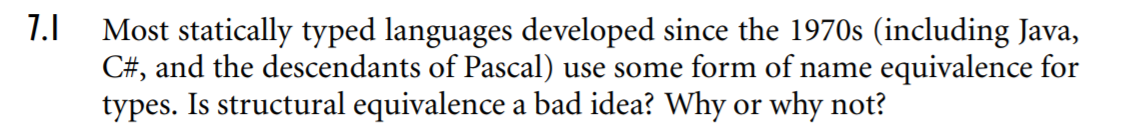
## NAME: Kostiantyn Makrasnov

**GRADE:**

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **POINTS** |  |
| EX07\_01 |  | 10 |
| EX07\_02 |  | 25 |
| EX07\_03 |  | 20 |
| EX07\_04 |  | 45 |
|  |  |  |
| **TOTAL** |  | 100 |

## EXERCISES:

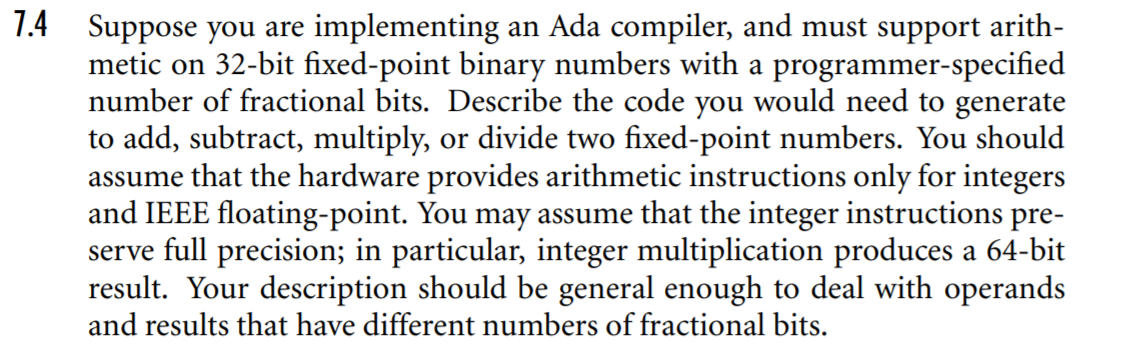
**EX07\_01 –** Complete Exercise 7.1, page 344

****

Noting the trend that most languages have moved on to name equivalence and recognizing the ambiguity that structural equivalence languages may pose, I believe that in most cases structural equivalence is a bad idea. When the types are checked by their names the compiler or interpreter forces the programmer to write more readable code as all variables that get assigned to each other will share the same type name. When structural equivalence is supported it may be confusing to some programmers why two variables of seemingly different types can be assigned to each other. Furthermore, if the ordering of the member variables inside the custom type doesn’t matter for structural equivalence (as in ML) it may be even more confusing/less readable as to why the assignment is allowed by the translator program.

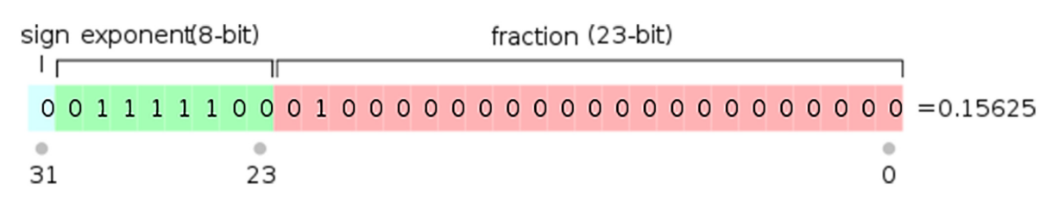
**References:**

1. <https://stackoverflow.com/questions/4486401/structural-equivalence-vs-name-equivalence>

**EX07\_02 –** Complete Exercise 7.4, page 344

[Answers on the next page]

32-bit fixed point binary numbers w/ programmer specified fractional bits:

Eg. 23 fractional bits:

**Add:**

1. add only the fractional portions of the numbers (pad with 0s at end as necessary to make both parts 32-bit integers), do so with built in operation take note if overflow occurred
2. add the exponent part of the two input numbers padding with 0’s this time at the start of number
3. add 1 to step 2 result if in step 1 an overflow was detected
4. copy the least number of bits needed to represent the exponent part from step 2 into the final number
5. fill the remaining bits with whatever fractional part fits into the 32-bit space, may lose some precision
6. carry the sign bit from step 3 to the final fixed-point value

**Subtract:**

1. Repeat all steps in the [add] procedure however change “add” to “subtract” operations and instead of looking for overflow in step 1 now we are looking for an underflow

**Multiply:**

1. Use the hardware integer multiplication operation to multiply together the exponent part of the input numbers
2. Use the hardware integer multiplication operation to multiply together the fractional part of the input numbers
3. Use the hardware integer multiplication operation to multiply together the fractional part of first input and exponent part of second input
4. Use the hardware integer multiplication operation to multiply together the fractional part of second input and exponent part of the first input
5. Add together the full result from step 1, respective exponent bits from part 3 and respective exponent bits from part 4
6. See if result in 5 fits into 32 bits with the 1st bit being the sign
   1. IF YES, add together step 2 result, fractional portion of part 3 and fractional portion of part 4, fit as much bits rom left to right into the result from step 6 to fill the 32 bit space
   2. If NO, cause an overflow and return a number with no fractional part

Note: here I obscure the process through which the compiler determines where the fractional bits start in steps 3, 4 (frankly because I do not know the math for that)

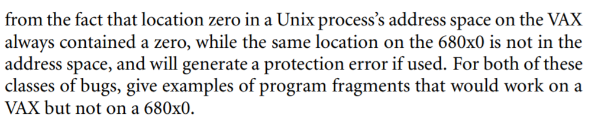
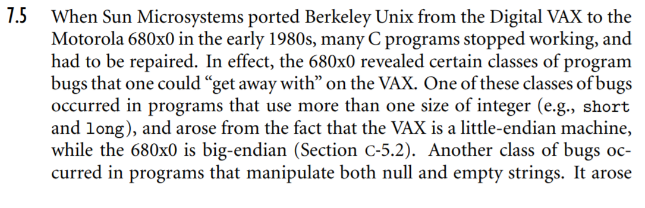
**Divide:**

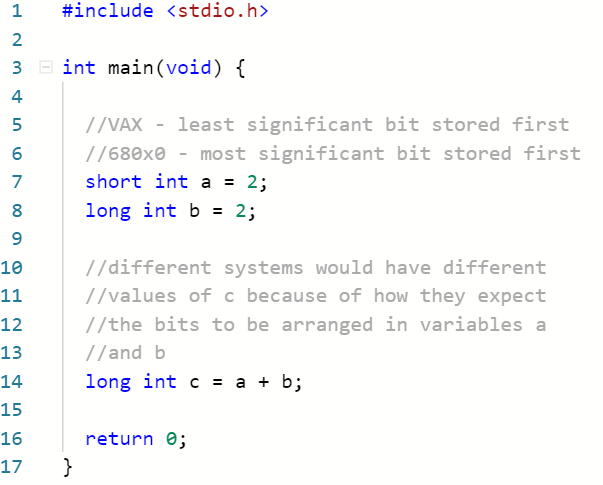
1. [I don’t know]

**References:**

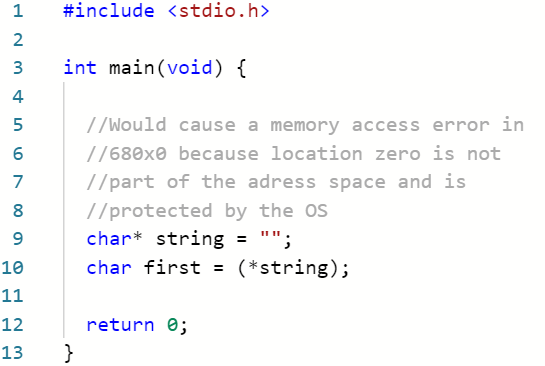
1. Graphic - <https://www.microcontrollertips.com/difference-between-fixed-and-floating-point/>

**EX07\_03 –** Complete Exercise 7.5, page 344-345



**Example of first bug:**

**Example of second bug:**



**EX07\_04 –** Complete Exercise 7.11, page 345-346

In your favorite language with generics, write code for simple versions of the following abstractions:

1. A stack, implemented as a linked list

**Done:** In project ./stack\_implementation/...

1. A priority queue, implemented as a skip list, a partially ordered tree (heap from data structures) embedded in an array

**Done:** In project ./priority\_queue\_implementation/...

1. A dictionary, implemented as a hash table

**Done:** In project ./dictionary\_implementation/...