Macquarie University
COMP 4000

Assignment 1 —

Using coupling invariants to calculate Super-Fibonacci numbers *very quickly*

For this assignment you must complete the corresponding Assignment 1 quiz, and submit an implementation in a single file

`Ass1COMP4000.java`

with the same information *and your name* appearing in the file itself.

Complete your answers as indicated by "Answer in iLearn Quiz" at right in the iLearn Quiz, and complete code stubs in the template file indicated by "Complete code stub".

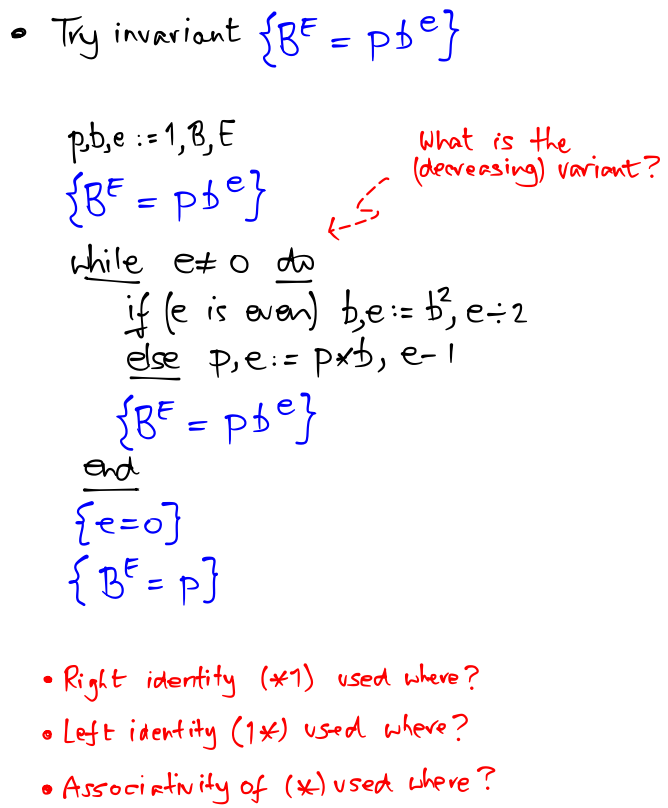
Please keep your answers short and precise whenever a written response is required.

*Do **not** use temporary variables in your submitted code.* Use simultaneous assignment statements, like the template code does. (Your editor will probably show syntax errors for the code but this will make it easier to mark. When you run your code of course you must make sure that syntax errors are resolved.)

You will be given a specific recurrence relation to work on in this assignment, available through the iLearn Quiz. You must use this recurrence relation in your answers and code from Task 3 onwards.

1 Task 1: High-school algebra helps to design your program

Questions 1–3 to be answered in the iLearn Quiz (5 marks)



One small detail for the matrix version is that the initialisation `p:= 1` must be replaced by `p:=` "the identity of matrix multiplication".

Figure 1: The program from lectures, in which variables p, b, B were numbers but could also have been 2×2 matrices.

In lectures we carefully worked out a program, repeated in Fig. 1, that calculates the E^{th} power B^E of a base B and places it in a variable p.

In that figure it turns out that there are four appeals to (three) elementary properties of multiplication: that 1 is a left- and a right identity of $(\times)$ and that $(\times)$ is associative. [1]

Recall that to find out what assertion must be true *before* an assignment statement such as `x,y,z:= X,Y,Z` in order to make an assertion $after(x, y, z)$ true after it, you simply make the substitutitions into $after$ that the assignment statement describes. In this case, what must hold before is therefore $\{after(X, Y, Z)\}$

Your first task is to answer three questions below about why we needed those arithmetic properties in the correctness argument for the program of Fig. 1. One sample answer is actually given for you, as an example of what you have to say, and of keeping it short and simple. Answer the other questions in the first three questions of the iLearn Quiz for this assignment.

1. We say that 1 is a *left identity* of $(\times)$ because $1\times x = x$ for all x. Why does 1 have to be a left-identity in order for $p, b, e{:=}\ 1, B, E$ to establish the invariant $\{B^E = pb^e\}$?

   ←Answer in iLearn Quiz (1).

2. We say that $(\times)$ is *associative* because $x\times(y\times z) = (x\times y)\times z$ for all x, y, z. Why does $(\times)$ have to be associative in order for assignment $b, e{:=}\ b^2, e\div 2$ to (re-)establish the invariant $\{B^E = p\times b^e\}$?(Note that associativity might not be the *only* property of arithmetic you need, and you can assume the others without mentioning them. But you *must* say where associativity is used.)

   ← This is an example question and answer.

   Answer: For $b, e{:=}\ b^2, e\div 2$ to re-establish $\{B^E = p\times b^e\}$ after its execution, we need $\{B^E = p(b^2)^{e\div 2}\}$ before its execution. We have the equality $(b^2)^{e\div 2} = b^{2\times(e\div 2)}$ from associativity, and then that $2\times(e\div 2) = e$ because e is even at that point in the program. [2]

3. Why does $(\times)$ have to be associative in order for $p, e{:=}\ pb, e-1$ to (re-)establish the invariant $\{B^E = pb^e\}$?

   ←Answer in iLearn Quiz (2).

[1] Please excuse the unavoidable variation in notations for multiplication: there is ordinary mathematical usage where x times y is written xy; then there is primary-school usage $x\times y$ for the same thing, when we want to be very explicit; and finally there is the programming-language usage $x*y$, because "×" is not an ASCII character.

[2] Note the two stages in this answer: first, use program logic, and *no* arithmetic. Then, use arithmetic, and *no* program logic.

4. Why does 1 have to be a right-identity of $(\times)$ in order for the postcondition $\{B^E = p\}$ to be true at the end of the program?

←Answer in iLearn Quiz (3).

2 Task 2: Transforming your program using a coupling invariant

Questions 4–6 to be answered in the iLearn Quiz, and one code stub to be completed. (10 marks)

We now assume that p,b and B in Fig. 1 are 2×2 matrices rather than simply natural numbers. Because *matrix* multiplication is associative and has a left- and right identity, just as normal multiplication of numbers does, the correctness argument still holds! [3] We will transform the program in Fig. 1 *in its matrix form* into one that just uses ordinary natural numbers to represent those matrices. The transformation uses the technique of coupling invariants.

Introduce twelve natural-number variables `B00`, `B01`, `B10`, `B11`, `b00`, `b01`, `b10`, `b11`, `p00`, `p01`, `p10` and `p11`. Then use the *coupling invariant*

$$B = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} \quad \text{and} \quad p = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix}$$

to rewrite the program in Fig. 1 as a Java program using Fig. 2 as a template. You are also asked to indicate the correct assignments to the B and p variables in iLearn.

←Answer in iLearn Quiz (4).

You will of course need to know how matrix multiplication works in order to do this. (There is abundant reference material available for elementary matrix algebra.)

Complete your code indicated in the file `Ass1COMP4000.java`:

← Complete code stub (1).

```
Mat fastMatrixExponentiation(int E)
// Pre E >= 0, B is this 2x2 matrix
//Post: Output the matrix equivalent to B**E
```

Recall that coupling invariants were the subject of the lectures in Weeks 5&6.

Do not *hand in the steps you went through to do the transformation — just give the program that the transformation produces. That is, your answer should be based precisely as the template in Fig. 2 except for your having*

[3] We also need distribution rules like $a(b{+}c) = ab{+}ac$ that hold for matrices as well. *Commutativity* is however not generally true for matrix multiplication.

replaced the question marks with expressions: give it the same structure, and use the same indentation. [4]

[4]This is not just to be fussy: it is to make marking more reliable. You can of course introduce linebreaks if necessary. But be neat!

```
// Variables B00,B01,B10,B11 and E are given.

var b00,b01,b10,b11: nat;
var p00,p01,p10,p11: nat;
var e: nat;

// Only B's should appear on the right here.
b00,b01,b10,b11:= ?,?,?,?;
// Only constants should appear on the right here.
p00,p01,p10,p11:= ?,?,?,?;
e:= E;

while (e!=0) {
    if (e%2==0) {
        // Only b's should appear on the right here.
        b00,b01,b10,b11:= ?,?,?,?;
        e:= e/2;
    } else {
        // Both p's and b's will appear here.
        p00,p01,p10,p11:= ?,?,?,?;
        e:= e-1;
    }
}
```

This program establishes $\begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}^E$.
(Also — please remove the comments from your answer: they are there only to give you hints about the transformation.)

Figure 2: Calculate the matrix-exponential very quickly. Use this as the basis for your Java implementation.

3 Task 3: Massaging your program

Questions 7–11 to be answered in the iLearn Quiz, and one code stub to be completed. (10 marks)

It turns out that we can use your matrix-exponentiation program from Task 2 to calculate the N^{th} Super Fibonacci number `SF(N)` very quickly and, when we do that, some of the program's variables can be removed. In this section we will discover which ones can be removed, and how to remove them.

The definition of a Super Fibonacci number is a generalisation of the original Fibonacci numbers, and are defined by s recurrence of the form:

$$SF(N{+}1) = a{\times}SF(N) + b{\times}SF(N{-}1) \ ,$$

where a, b are integers and $SF(0) = 0$ and $SF(1) = 1$.

The key fact is that we have

$$\begin{pmatrix} SF(N{+}1) \\ SF(N) \end{pmatrix} \quad = \quad \begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix}^N \begin{pmatrix} 1 \\ 0 \end{pmatrix} \ ,$$

which you can verify for yourself by doing the matrix multiplication by hand for small N's like $0, 1, 2 \ldots$. What our matrix-exponentiation program does is to calculate that N^{th} power on the right, i.e. for the general case. We just have to

- Supply $\begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix}$ for B, where a, b are integers
- Add a final assignment

  $f{:=}$ "the lower element of the $2{\times}1$ matrix $p \times \begin{pmatrix} 1 \\ 0 \end{pmatrix}$" .

Given our coupling invariant, the last of those three means adding a final assignment `f:= p10`; and the first of the three means we can replace the `B`'s by constants.

Now we remove some variables.

In iLearn you are given specific values for a and b in the form of a recurrence in Q10. You must use the values given in all of your answers and programs below wherever it is relevant (i.e. all questions and code asked for in Tasks 3 and 4). For example if your recurrence in Q10 is:

$$SF(N+1) = 3 \times SF(N) + 6 \times SF(N-1)$$

then you must use 3 for a and 6 for b.

3.1 Two auxiliary variables can be removed

In this section you'll figure out which variables in your implementation can be considered to be auxiliary.

In the final assignment to `f` proposed just above and to be added to your version of Fig. 2, only one of the four `p`'s appears on the right-hand side. Indicate which one. ←Answer in iLearn Quiz (7).

Indicate which pair (i.e. not all three) of the other `p`-variables can be considered auxiliary in your program if `f` is to be the only output. ←Answer in iLearn Quiz (8).

←Answer in iLearn Quiz (9).

3.2 An extra invariant allows two more variables to be removed

In our special case where the input B is set to

$$B \quad = \quad \begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix} \tag{1}$$

in order to calculate Super Fibonacci numbers, the `b`-variables you introduced in Task 2 will turn out to satisfy the (extra) invariants

$$b_{00} = a \times b_{10} + b_{11} \quad \text{and} \quad b_{01} = b \times b_{10} \tag{2}$$

during the program's execution. We say "extra" invariants because they are not necessary to prove the program is correct: but they are invariants nevertheless.

For your recurrence relation given in Question 10 in the iLearn quiz, select the invariant conditions.

Now use those invariants to remove variables `b00` and `b01` completely from your program. Using cut-paste-edit as before, give your new, smaller program. Its variables should be (only) `b10`, `b11`, `p10`, `p11`, `e`, `f` and `N`. ←Answer in iLearn Quiz (10).

Confirm which pair of b-variables are considered auxiliary from the original program. ←Answer in iLearn Quiz (10).

Complete the code stub in the file Ass1COMP4000.java: ←Answer in iLearn Quiz (11).

```
int  fastSuperFibonacci(int N)
// Pre: N >= 0, B is this 2x2 matrix;
//Post: Output the N'th Super Fibonacci number
//      determined by your recurrence SF(N+1)= a*SF(N) + b*SF(N-1)
```

←Complete code stub (2).

4 Task 4: Running your program

Question 12 to be answered in the iLearn Quiz (5 marks)

The program in Fig. 3 uses the slow, "obvious" algorithm to calculate the six least-significant digits of `SF(N)` in time *linear* in N. It is strongly suggested that you test it out for yourselves; but *do not hand-in your tests of this program.*

In this part of the assignment you are asked to compute your Super Fibonacci number for a very large value of N. Since the value you are asked to compute is likely too big to be evaluated without an overflow error, you are asked to produce only the *last six digits* of your Super Fibonacci number. For this you can use modular arithmetic. When you run your program you should adjust the arithmetic multiplications and additions so that they use modular arithmetic. The lecture in Week 6 will explain how to do this.

Take your student number (say 23456789) and retain only its first six digits (e.g. 234567). Make a new number by repeating those six digits 3 times (e.g. 234567234567234567). Run your fast Super Fibonacci program, (adjusted for modular arithmetic):

```
fastSuperFibonacci(int N)
```

with this large number as input for `E`, and cut-and-paste the terminal output as your answer for this section. (For student id 23456789, and $a = b = 1$we would expect to see

```
The low-order digits of SF(234567234567234567) are 498978.
%
```

as the answer for this part.)

←Answer in iLearn Quiz (12).

You might want to test it by comparing its output with `slowFib...` at least for smallish values of N.

Finally, for a bonus mark, give an estimate of how long the slow version of the exponentiation would take to do the same calculation you just did with the fast version, and justify your estimate.

On a platform that supports unlimited-precision integer arithmetic (e.g. *Python*) you can run your (transcribed) programs without the modular arithmetic and try printing the whole answer, at least for smallish values of N.

```
long DIGITS= 1000000;
long this= 0;
long next= 1;
long n;
for (n= 0; n!=N; n++) {
    long thisN= next;
    next= (b*next+a*this) %DIGITS;
    this= thisN;
}
```

Use this as the basis for computing the last six digits of the N'th Super Fibonacci number defined by the recurrence

$$SF(N+1) = a \times SF(N) + b \times SF(N-1) \ .$$

When your program successfully compiles (after fixing any errors(!)), you can run it with for example `./a.out 9999` and it should print the last six digits of the 9999^{th} Super Fibonacci number with $a = b = 1$. (They are 230626 :-)

Figure 3: Calculate the last six digits of `SF(N)` quite slowly.

An interesting question (but not part of the assignment) is

Roughly speaking, how many decimal digits does the N^{th} Super Fibonacci number have?