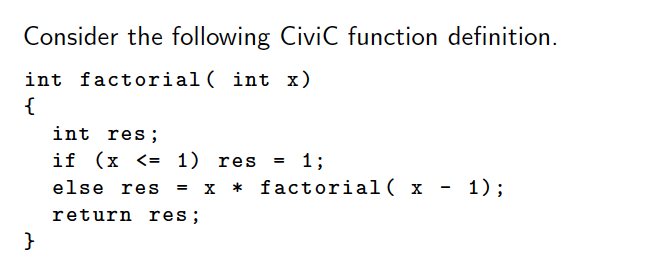
Assignment 6

Andrea & Aynel



**Assignment 21: Code generation**

[a] Manually generate CiviC-VM assembly code for the above function definition making use of labels to mark destinations of jump instructions.

iload 0 // load x

iloadc 0 // load 1

ile // if x <= 1

branch\_f **L1** // go into if statement

istore 1 // res = 1;

jump **L2** // skip else statement

**L1:** // else

iload 0 // load x

iloadc 0 // load 1

isub // x – 1

functioncall SOMETHING

iload 0 // load x

imul

istore 1 // res = x \* factorial ( x – 1)

**L2:** // return res

>> SOMETHING

[b] Point out the relationship between assembly code and source code through line comments in the assembly code.

See above.

[c] Add the number of bytes required for each line of CiviC-VM assembly code. Assume here jump instructions would take byte code offsets as arguments and not labels.

[d] Compute the proper byte code offset for each jump instruction; consult the CiviC-VM manual for details on individual instructions.

**Assignment 22: Compilation Schemes Revisited**

Devise a compilation scheme that replaces each occurrence of a for-loop in the body of a CiviC function by semantically equivalent CiviC code that makes use of a while-loop instead. As a simplication consider only for-loops without a step specication and assume that CiviC would support arbitrary interleaving of variable declarations and statements in function bodies following the example of C99.