Formal Languages & Compilers
Prof. Breveglieri
Exam – Laboratory exercise[1]
09/09/2005

SURNAME and FIRST NAME:..... Student ID:.....
Program: ○ Laurea Specialistica ○ Vecchio Ordinamento ○ Other: .....

Given the SimpleVM machine and the Simple compiler attached to the exercise, consider the following SimpleAssembler code. It is partially commented, and includes function calls:

```
    jmp    main

                 /* Begin function code */
X:  ldvar    -2   /* -2 is the "name" of the second input parameter */
    ldint     0
    gt
    jmp_false Y
    ld_var   -3
    ld_int    0   /* Reserve space for return value */
    ld_var   -3   /* -3 is the "name" of the first input parameter */
    ld_var   -2
    ld_int    1
    sub
    call      X   /* Recursive function call */
    data     -2   /* Pop the first two elements of the stack */
    add
    store    -4   /* -4 = return value */
    ret
Y:  ld_var   -3
    store    -4
    ret           /* End function code */

main: data    1   /* Main function code */
    readint   0
    readint   1
    ld_int    0   /* Reserve space for return value */
    ld_var    0   /* First function parameter *
```

[1]Time: 30'. You can use books and personal notes. You can use pencils. If additional sheets are needed, write your name on each of them. *Please reply to each point individually!*

```
ld_var    1   /* Second function parameter */
call      X   /* Function call */
data     -2   /* Pop the first two elements of the stack */
writeint
halt
```

The semantics of the `call` and `ret` instructions is as follows.

call $< to >$ Saves on the top element of the stack the address value and the AR register (i.e., the caller's *Activation Record* pointer), in that order (first PC, then AR). Then sets the PC to the $< to >$ value and AR to the new top element of the stack.

ret AR=stack[top]. PC=stack[top-1]. top=top-2.

Given this information, answer the following points:

1. Design syntactic constructs in the Simple source language that provide function declaration and call, compatible with the proposed bytecode.

2. Write the source code corresponding to the proposed bytecode.

3. Modify the Flex and Bison specifications of Simple, adding the rules necessary to parse the new constructs.

4. Propose a translation schema and implement it in the Simple compiler, using Bison semantic actions.

5. Modify the SimpleVM interpreter to correctly handle `call`, `ret`.

6. **bonus**: add in the Bison specification at least a semantic check during the compilation (e.g.: number of formal parameters = numbers of actual parameters).