Work Time: 2 hours and 50min
Please implement one problem in Java and the other problem in C#.
You are free to choose which problem will be implemented in Java and which problem will be implemented in C#.
If a problem implementation does not compile or does not run you will get 0 points for that problem (that means no default points)!!!
If for one problem you have only a text interface to display the program execution you are penalized with 1 point for that problem. The GUI to input the ToyLanguage programs is not necessary, you can hard code the input programs in your implementations.

1. **(0.5p by default). Problem 1:Implement procedures in ToyLanguage.**
   a. **(0.75p).** Modify the program state such that each program state has a stack of SymTables instead of only one SymTable. The program state runs using the top of the stack as the current SymTable. All the previous expressions and statements must support this modification. The fork statement execution must clone the entire stack of S.
   b. **(0.5p).** Inside PrgState, define a new global table (global means it is similar to Heap and Out tables and it is shared among different threads), ProcTable that maps a procedure name to a pair of a list of variable names and a statement. The list of the variable names are the formal parameters of that procedure, while the statement is the procedure body. ProcTable must be supported by all the previous statements. The procedures cannot be overloaded, that means one procedure name has only one set of formal parameters and body. It must be implemented in the same manner as Heap, namely an interface and the class which implements the interface.
   c. **(0.25p).** Before to input the main program you must allow the user to introduce the procedures. A procedure is defined as follows: procedure proc_name(v1,...,vn) statement where proc_name is the procedure name, v1...vn is the list of variables names (formal parameters) and the statement is the procedure body.
   d.(0.75p). Define the new statement call fname(exp1,...,expn) that represents a the call by value of the procedure fname defined in the ProcTable. The execution of this statement is the following:
      - pop call fname(exp1,...,expn) from the ExeStack
      - check ProcTable and extract the list of the variables v1,..vn and the body statement for the procedure fname. If the procedure fname does not exist in ProcTable print an error message and terminate the execution of the program.

- use the current SymTable and evaluate the current arguments of the procedure, namely eval(exp1), ..., eval(expn)
- create a new SymTable that contains the mappings of the formal variables vi to the value of the current arguments expi (namely vi->eval(expi))
- push new SymTable on the stack of SymTables
- push the statement return on the ExeStack (statement return is defined below)
- push the body of the procedure fname on the ExeStack

e. (0.5p). Define the new statement

return

that is automatically placed on the ExeStack by the execution of a call statement. The statement return is not used in the programs of our toy language. The execution of this statement is the following:

- pop return from the ExeStack
- pop the top of the stack of SymTable (namely restore the local variables of the caller)

f. (0.5p). Extend your GUI to suport step-by-step execution of the new added features. It is not necesary to have a GUI to input the ToyLanguage programs, therefore please hard code the examples to be run by your implementation.

e. (1.25p). Show the step-by-step execution of the following program. At each step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a text readable log file.

The following program must be hard coded in your implementation.

```
   procedure sum(a,b)  v=a+b;print(v)
procedure product(a,b) v=a*b;print(v)
   and the main program is
       v=2;w=5;call sum(v*10,w);print(v);
fork(call product(v,w));
       fork(call sum(v,w))
```

The final Out should be {25,2,10,7}

2. **(0.5p by default) Implement Sleep statement in Toy Language.**

a. (2.25p). Define the new statement: sleep(number)

Its execution on the ExeStack is the following:

- pop the statement
- if number== 0 then do nothing else push sleep(number-1) on the stack

b. **(1p).** Extend your GUI to suport step-by-step execution of the new added features. It is not necesary to have a GUI to input the ToyLanguage programs, therefore please hard code the examples to be run by your implementation.

c. **(1.25p).** Show the step-by-step execution of the following program. At each

step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a text readable log file.

The following program must be hard coded in your implementation:

v=0;
(while(v<3)
(fork(print(v);v=v+1);v=v+1); sleep(5);
print(v*10)

The final Out should be {0,1,2,30}