**Homework 5 - Register allocation**

In this stage of the project you will translate the Spiglet code to an even lower level intermediate, Kanga. Kanga resembles Spiglet but is closer to the specifics of the MIPS architecture. Changes from Spiglet to Kanga are as follows:

* Kanga supports C-like comments
* Labels are now full, global symbolic addresses, not just in local scope
* Instead of an unlimited number of temporary local variables, Kanga has 24 registers of global scope. Registers s0-s7 and t0-t9 can be used for anything. Registers a0-a3 are used only for passing arguments to a function call. The v0 register must contain the result of a function call, and v0 (again) and v1 will be used as temporary registers when you need to perform operations on stack-allocated values.
* The values are loaded and written to the stack with the commands ASTORE and ALOAD, where the expression (SPILLEDARG i) refers to the i-th value on the stack, with the first value at (SPILLEDARG 0). For example the operation "ALOAD s3 (SPILLEDARG 1)" loads the second value from the stack into the s3 register.
* The body of a function is no longer a StmtExp but a StmtList. The return value must be in register v0.
* CALL is a now a statement and not an expression. As mentioned above, "a" registers are used for arguments. If there are more than 4 arguments you must use the PASSARG command, which saves additional arguments on the stack. The numbering of PASSARG starts from 1, while the numbering of SPILLEDARG starts from 0, so in general, an argument that is passed as (PASSARG i) is visible in the body of the function as (SPILLEDARG i-1). For example, imagine that we call a function with label P with the values of arguments in registers t1, t2, t3, t4 and t5, and we want the return value to be stored in register t6:

|  |
| --- |
| * + MOVE a0 t1 // first put the values in argument registers   + MOVE a1 t2   + MOVE a2 t3   + MOVE a3 t4   + PASSARG 1 t5 // more than 4 arguments placed in stack   + CALL P   + MOVE t6 v0 // return value |

* A function has three numbers at the beginning of its definition, e.g. "procA [5] [3] [4]". The first number has the same meaning as in Spiglet, i.e., it represents the number of arguments. The second is the number of stack locations needed by the function. This is the total number, which includes space for arguments, if needed, as well as for any local variables not stored exclusively in registers, and for any registers that must be saved. The third number is the maximum number of passed arguments in calls appearing in the procA method body. If e.g. procA calls procB, which takes 3 arguments, procC, which takes 2, and procD, which takes 4 arguments, then this number is 4.

Here you will find the [Kanga](http://cgi.di.uoa.gr/%7Ethp06/project_files/kanga.html) grammar (also in [JavaCC](http://cgi.di.uoa.gr/%7Ethp06/project_files/kanga.jj) form), our usual examples of programs in [Kanga form](http://cgi.di.uoa.gr/%7Ethp06/project_files/kanga-examples) and an [interpreter for Kanga](http://cgi.di.uoa.gr/%7Ethp06/project_files/kgi.jar). Finally, you must provide a short README that describes which register allocation algorithm you chose, and gives a high-level overview of the implementation (e.g., classes responsible for the register allocation, code conventions, etc.).

Your program should run as follows:  
  
java [MainClassName] [file1.spg] [file2.spg] ... [fileN.spg]  
  
That is, your program must compile to Spiglet all **.spg** files given as arguments. Moreover, the outputs must be stored in files named file1.kg, file2.kg, ... fileN.kg respectively.