C SC 230 – Summer 2014 – Assignment 4

Due Friday Aug. 1 by 15:00 in the course box in ECS, but will be accepted (without penalty) until Tuesday Aug. 5 at 13:00

Total Marks = 50

PART 1: The Problem to be solved: the Kolakoski Sequence

The Kolakoski sequence (A006928) is an infinite list whose initial 108 entries are shown in Table 1.

Table 1: The first 108 entries of the Kolakoski sequence A006928

elements 1 to 18	1	2	1	1	2	1	2	2	1	2	2	1	1	2	1	1	2	2
elements 19 to 36	1	2	1	1	2	1	2	2	1	1	2	1	1	2	1	2	2	1
elements 37 to 54	2	2	1	1	2	1	2	2	1	2	1	1	2	1	1	2	2	1
elements 55 to 72	2	2	1	1	2	1	2	2	1	2	2	1	1	2	1	1	2	1
elements 73 to 90	2	2	1	2	1	1	2	2	1	2	2	1	1	2	1	2	2	1
elements 91 to 108	2	2	1	1	2	1	1	2	2	1	2	1	1	2	1	2	2	1

The only numbers allowed are 1 and 2. Each number states how many numbers need to be appended to the sequence. There can be no more than two of the same number in sequence. Every time a new number is considered (read), one must alternate between writing 1 and 2. A recurrence plot of the Kolakoski sequence is illustrated in Figure 1 and its point plot in Figure 2.

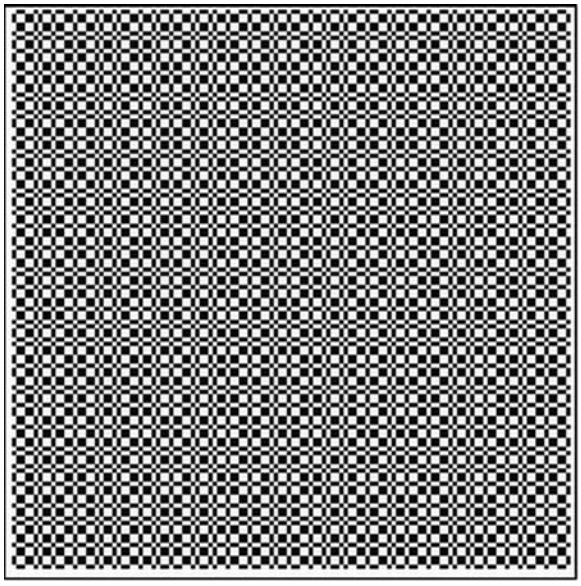

Figure 1: Recurrence plot

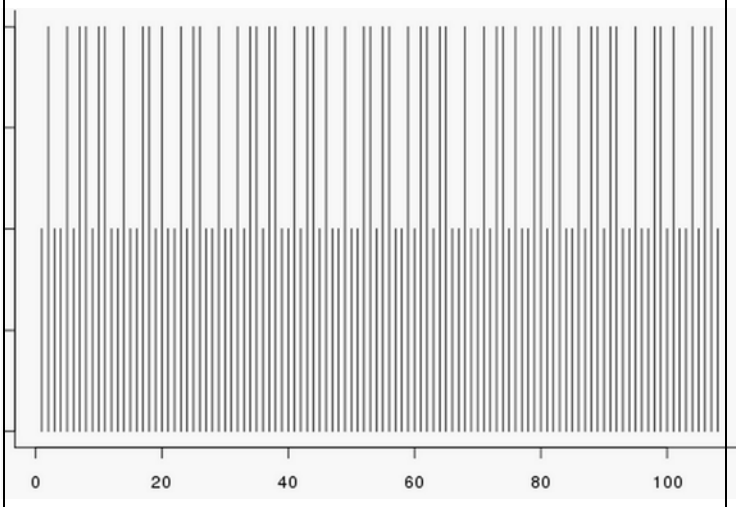

Figure 2: Point plot

There are actually a few permutations of the Kolakoski sequence (with different number labels).

The easiest english explanation is found for the development of the Kolakoski sequence A000002, shown by example in Table 2.

Table 2: Example for Kolakoski sequence A000002

write 1;	==> [**1**]
read 1 as the number of 1's to write before switching to 2;	==> [**1**]
write 2;	==> [**1 2**]
read 2 as the number of 2's to write before switching back to 1;	==> [**1 2 2**]
read the new 2 as the number of 1's to write;	==> [**1 2 2 1 1**]
read the new 1,1 as stating one needs to write one 2's and one 1's	==> [**1 2 2 1 1 2 1**]
continue generating forever.	

Kolakoski's self-generating sequences are of interest to both computer scientists and mathematicians (see "*A New Kind of Science*", p. 895, by Stephen Wolfram). One might expect that the density of 1's and 2's is 50% for each, but this conjecture has never been formally proved (see the On-Line Encyclopedia of Integer Sequences™ (OEIS™) at http://oeis.org/Seis.html). Further references can be found in N. J. A. Sloane and Simon Plouffe, "*The Encyclopedia of Integer Sequences*", Academic Press, 1995.

1.1 Algorithm, PseudoCode and Output

While the steps in Table 2 appear to be a simple enough explanation, it is still far from being a proper algorithm ready for programming. It would be a great learning experience to leave the development of the algorithm and pseudo code as part of this assignment, but it may take away from the real focus, which is to program in both C and ARm and link the code together.

Thus the pseudo code is given to you in Figure 3 for the production of the Kolakoski sequence A006928 for which a closed form algorithm can be found (not for the one shown in Table 2 which is the A000002 sequence used only for explanation) . Please note carefully the language (in case you do not have much experience). When it is stated that the range of a variable is "from x to y" this implies that both x and y are included in the possible values.

```
Let Kseq be the the the array to contain the Kolakoski sequence
Initialize the beginning elements as:                          } in main
   Kseq[0] is not used
   Kseq[1] = 1
   Kseq[2] = 2
Let n be the number of "run" steps in producing the sequence
where n ranges from 2 to infinity. In a practical program, n should range
from 2 to a given maximum number of "runs", here denoted by MAXRUNS

   for (n=2 to MAXRUNS)                                        } in
   {                                                           } function
      for(i=1 to i= Kseq[n])                                   } AppendKseq
      {
         append the element (1+(n mod 2)) to Kseq
      }
   }
```

Figure 3: Pseudo code for producing the Kolakoski sequence

The output of a program should be the list of entries of the Kolakoski sequence up to a maximum number of runs, together with a count of the total number of entries, the count of the number of elements equal to 1 and the corresponding count of the number of elements equal to 2. A sample output for a given maximum run of n=50 is shown in Figure 4, where elements are printed 10 per row.

```
Jean Luc Picard V00123456
Kolakoski Program starts

Kolakoski sequence of length 75 with 50 Runs
Number of 1's is = 38
Number of 2's is = 37
      1    2    1    1    2    1    2    2    1    2
      2    1    1    2    1    1    2    2    1    2
      1    1    2    1    2    2    1    1    2    1
      1    2    1    2    2    1    2    2    1    1
      2    1    2    2    1    2    1    1    2    1
      1    2    2    1    2    2    1    1    2    1
      2    2    1    2    2    1    1    2    1    1
      2    1    2    2    1
Kolakoski Program ends
```

Figure 4: Sample output

Make sure your program can work properly even if the maximum number of runs (`MAXRUNS`) is a different number, as, in marking, we may change your code and run it again with different values! **DO NOT**, however, make `MAXRUNS` a variable input from the user, or from a command line, or a parameter to a function (instead use `EQU` and `#define`).

1.2 The program design

The structure of the program must include at least two functions called by "`main`", namely: "`AppendKseq`" and "`PrintKseq`". Do not forget to add appropriate messages at the beginning and at the end of your program, and whenever output is expected. The interface of these two functions should be as follows (no changes allowed):

AppendKseq. `int AppendKseq(int Kseq[],int startindex,int *Num1)`
Description: Append the next elements in the Kolakoski sequence to the initial array.
Inputs: address of initial `Kseq` array, `index` = next position to be filled; `Num1` = address of variable for the number of entries equal to 1.
Output: `int`= next position to be filled.
Side effects: update number of entries equal to 1 in its variable `Num`.
Expectation: `Kseq` array has been initialized for the first 3 entries at least (i.e. `index` > 2)

PrintKseq. `void PrintKseq(int Kseq[],int index)`
Description: Print the Kolakoski sequence.
Inputs: address of `Kseq` array, `index` = next position to be filled.
Output: none.
Side effects: none. However the elements of `Kseq` are printed in rows, possibly 10 elements per row.
Expectation: `Kseq` is not empty.

MAXRUNS. Use a `#define` in C and an `.EQU` in ARM.

Maximum size of Kseq array. It should be `MAXRUNS*2` in C and similarly, in bytes for ARM.

PART 2: The implementation using C

Following your design from a flowchart or precise pseudo-code (do it!) implement your program using C, in a file called "A4KolakoskiC.c", compiled using:

```
gcc -Wall A4KolakoskiC.c
```

Test your code for any number of MAXRUNS from 10 to 100 simply by recompiling it with different initial values (you must not read values from a file or from the command line or from stdin). Submit the program with a pre-encoded value of MAXRUNS=50.

PART 3: The second implementation using ARM

Implement your program using ARM, in a file called "A4KolakoskiA.s", tested using ARMSim#, for any number of MAXRUNS from 10 to 100 simply by reassembling it with different initial values. Submit the program with a pre-encoded value of MAXRUNS=50. Important to note at this point is that in the next deliverable you will substitute the ARM function AppendKseq with the C function. Thus, before you code the ARM one, make sure that the parameter passing in registers follow the C expectations of the compiler, otherwise you may find yourself rewriting code later on! Read below before doing this part.

PART 4: The fourth implementation using both C and ARM

Finally write the last version of the Kolakoski program using a mixture of C and ARM. Use the instructions posted on the website to combine multiple object files and to make sure that all parameters are appropriate (be careful about what the compiled C code may change!). Note carefully: the goal here is that you do not write any new code. You should be able to insert your original C code for AppendKseq in a separate file called by the ARM program. We will check if any changes were made between the two versions! The trick is that the function should be called in ARM passing the parameters as expected by the C compiler.

Use the following template with functions and procedures as indicated.

```
Main: in ARM assembly, in a file A4KolakoskiAC.s.
AppendKseq: in C, in a file A4AppendKseqCsub.c.
PrintKseq: in ARM assembly, in the same file as main.
Any other function: in ARM assembly, in the same file as main.
```

Basically you are repeating the second implementation while substituting the C code for AppendKseq in an external C source file, A4AppendKseqCsub.c, containing the C source code for AppendKseq, which needs to be cross compiled to ARM code. Then ARMSim# will be able to link it in with the assembled A4KolakoskiAC.s and the program should run as it did both in the second full ARM implementation and the third one with multiple files.

PART 5: What to submit: all the deliverables

1. Submit the source code for:
   - "A4KolakoskiC.c" (from implementation 1).
   - "A4KolakoskiA.s" (from implementation 2).
   - "A4KolakoskiAC.s" and "A4AppendKseqCsub.c" (from implementation 3).

5.1 References

- A web calculator for the Kolakoski sequence and others in general:
  http://members.chello.nl/k.ijntema/kolakoski.html
  http://members.chello.nl/k.ijntema/
- Wolfram MathWorld:
  http://mathworld.wolfram.com/
- The On-Line Encyclopedia of Integer Sequences™ (OEIS™): use the A006928 sequence
  http://oeis.org/Seis.html