Problem 2: RandoHMMM Numbers | Homework 7 | Contenu du cours CS005x

Problem 2: PseudoRandoHMMM Number Generation

In this problem, you'll create a new HMMM program that generates pseudorandom numbers.

Here is some starter code:

```
00 read r1    # input a
01 read r2    # input c
02 read r3    # input m
03 read r4    # input X_0
04 read r5    # input N
```

The next sections first explain these inputs, and then explain how to generate pseudorandom numbers with them.

The Math Behind (Pseudo-)Random Number Generation

A linear congruential generator (LCG) is a "pseudorandom-number generator" algorithm that generates a sequence of numbers that "look and feel" like random numbers. The numbers generated are not truly random because they are generated by a mathematical formula, but they have statistical properties that make them behave like true random numbers.

The LCG algorithm is defined by the recurrence relation:

$X_{n+1} = (a\ X_n + c)\ \%\ m$

where:

- m is a divisor (whose remainder is preserved)
- a is a multiplier
- c is an increment
- X_0 is a "seed value," which is between 0 and m (excluding m)

Typically, the user is asked to enter the seed value, X_0.

An LCG random-number generator then uses the above formula to compute X_1, which is the first "pseudorandom" number.

After that, X_2 is computed from X_1, and so on forever (or until we have enough pseudorandom numbers for our needs!).

Notice that the pseudorandom numbers generated this way are always between 0 and $m-1$ because we are "modding" our numbers by m. Mod already exists in HMMM (you don't need to write it!).

Since the sequence of numbers produced depends only on X_0 and the generator's parameters, the maximum period (how many numbers are generated before the sequence repeats) of the LCG is at most m (why?). If the LCG actually

has period `m`, then it is said to have **full period**. This is a desirable property for a random number generator since it means that it generates many different numbers before repeating!

Part 1: Writing the RandoHMMM Number Generator

Your first job is to implement the LCG algorithm in HMMM! Your program should work as follows: the user will input **five** values in the following order (please use this order, since your program will be graded by providing inputs in the same order):

- First, the user enters the number `a`, the multiplier in the LCG algorithm.
- Second, the user enters the number `c`, the increment in the LCG algorithm.
- Third, the user enters the number `m`, the modulus divisor in the LCG algorithm.
- Fourth, the user enters the seed, X_0, in the LCG algorithm.
- Fifth, the user enters a number `N`, indicating the number of pseudorandom numbers that should be printed.

The HMMM program should then print the `N` pseudorandom numbers, beginning with X_1 (X_0 is not considered one of the pseudorandom numbers and is not printed.)

Extend the `random` starter code provided above to the full random-number generator as just described.

Note that **`mod`** is built-in to HMMM! **Don't write mod yourself!**

You will find you need to copy one register into another. The `copy r4 r8` command copies the contents of register `r8` **into** register `r4`.

Caution! the `copy` command is right-to-left!

Checking your generator

To check that your random-number generator is working, try running it with the following inputs:

- First, enter the number `a = 10`, the multiplier in the LCG algorithm.
- Second, enter the number `c = 7`, the increment in the LCG algorithm.
- Third, enter the number `m = 11`, the modulus in the LCG algorithm.
- Fourth, enter the seed, X_0 `= 3`, the starting value in the LCG algorithm.
- Fifth, enter the number `N = 10`, indicating that 10 pseudorandom numbers should be printed.

The output should then be ten alternating `4`s and `3`s:

```
4
3
4
3
4
3
4
3
4
```

```
3
```

Clearly, these are not good values for our random-number generator—they are not very "random"! The next section will ask you to choose much better values.

Part 2: Picking Parameter Values for the LCG

In this part you will choose "good" values for the parameters `a`, `c`, and `m` in the LCG algorithm. You should do this *by hand*, guided by the constraints below:

It turns out that the LCG algorithm has its best-possible performance—that is, it generates `m` different values before repeating—if the following three conditions are met:

- Condition 1: `c` and `m` are relatively prime (that is, `c` and `m` have no common divisors other than 1)
- Condition 2: (`a`-1) is divisible by all *prime factors* of `m` (not *all* factors of `m`, all *prime factors* of `m`)
- Condition 3: (`a`-1) must be a multiple of 4 if `m` is a multiple of 4

Your boss at SPRANG Corp. (Spam-Processed RAndom Number Generation) has asked you to construct a random number generator with `m` equal to 100. Find the smallest values of `a` and `c` that can be used with this value of `m` and satisfy these three conditions.

Place a comment at this point of your code indicating the values that you found for `a` and `c`.