Alexander Ryzhko
APPLICATION SECURITY CS9163
ASSIGNMENT 1.1

The file format:

- instruction must be taken from a text file
- to execute in ubuntu cd terminal to the folder with thesanbax.py and instruction file
    - python thesandbox.py instructionFile
- Instructions line must start with an instruction.
    - Instruction line must not be longer than 60 characters
    - There must be less than 10000 instructions in the instruction file
- After instruction there must be a space followed by needed arguments separated by space space
- There must be no empty lines. Starting from first line of the file if the sandbox reads an empty line it stops executing and exits as if there is no more instructions to execute
- There are 27 variables to use:
    - 26 variables are lower case (case sensitive) alphabet characters initialized to 0 are free to use by user
    - 1 variable is upper case R:
        - this variable stores result after some computational instruction execution
    - all variables are integer type
        - user must not exit integer limits:
            - max = 2147483647
            - min = -2147483648
    - user can declare additional variables
        - must not be a number,
        - must not contain ‘.’,
        - must not be already declared
        - must not be longer than 7 characters
        - total variable amount must not be larger than 10000
- There are following instructions (case sensitive):
    - EQL - equal equal instructions (similar to ==)
        - must have 2 argument followed
        - stores result of operation in variable R
            - 1 - if 1st argument is equal to 2nd argument
            - 0 - otherwise

    - NEQ - inverse of EQL (similar to !=)
        - must have 2 argument followed
        - stores result of operation in variable R

            - 1 - if 1st argument is not equal to 2nd argument
            - 0 - otherwise
    - GRE - greater than (similar to >)
        - must have 2 argument followed
        - stores result of operation in variable R
            - 1 - if 1st argument is greater than 2nd argument
            - 0 - otherwise
    - LES - less than (similar to >)
        - must have 2 argument followed
        - stores result of operation in variable R
            - 1 - if 1st argument is less than 2nd argument
            - 0 - otherwise
    - GRQ - greater than or equal to (similar to >=)
        - must have 2 argument followed
        - stores result of operation in variable R
            - 1 - if 1st argument is greater than or equal to 2nd argument
            - 0 - otherwise
    - LEQ - less than or equal to (similar to <=)
        - must have 2 argument followed
        - stores result of operation in variable R
            - 1 - if 1st argument is less than or equal to 2nd argument
            - 0 - otherwise
    - ADD - addition
        - must have 2 argument followed
        - stores result of operation in variable R
            - sum of 1st argument with 2nd argument
    - SUB - subtraction
        - must have 2 argument followed
        - stores result of operation in variable R
            - difference between 1st argument and 2nd argument
                - 1st - 2nd
    - MUL - multiplication
        - must have 2 argument followed
        - stores result of operation in variable R
            - product of 1st argument with 2nd argument

    - DEV - division
        - must have 2 argument followed
        - stores result of operation in variable R
            - quotient of 1st argument with 2nd argument
    - MOD - modular

        - must have 2 argument followed
        - stores result of operation in variable R
            - remainder of quotient of 1st argument with 2nd argument
    - AND - bitwise AND operator
        - must have 2 argument followed
        - stores result of operation in variable R
    - OR - bitwise OR operator
        - must have 2 argument followed
        - stores result of operation in variable R
    - INV - bitwise inverter
        - must have 1 argument followed
        - stores result of operation in variable R
    - NOT - logical inverter
        - must have 1 argument followed
        - stores result of operation in variable R
            - 1 - if argument = 0
            - 0 otherwise
    - ASS - assignment
        - must have 2 argument followed
            - 2nd argument must be a variable
        - stores value of first argument into 2nd argument
    - LOG - outputs to the standard output (screen) values of the given arguments
        - must have at least 1 argument
    - DEC - declares new variable
        - must have 1 argument -> var name; initialized to 0
        - must not be a number,
        - must not contain ‘.’,
        - must not be already declared
        - must not be longer than 7 characters

    - JUM - go to on the condition
        - after this instruction space and one of the following condition instructions must follow: EQL,NEQ,GRE,LES,GRQ,LEQ
        - after a condition instruction 1 or 2 argument must follow
            - 1st argument is the line where is execution should go to if condition holds true
                - if only 1 argument given and condition is false execution will

continue on the next line
    - 2nd argument is the line where execution should go to if condition is false
- Example 1 count:
    - ASS 10 x <- assigning value 10 to a variable x
    - LOG x <- output value of variable x on the screen
    - SUB x 1 <- subtracting 1 from variable x;result stores in variable R
    - ASS R x <- assigning value of variable R to variable x
    - JUM GRE x 0 2 <- if x is greater than 0 go to line 2 (LOG x)
- Example 2 fibonacci:
    - ASS 10 f <- assigning value 10 to variable f
    - ASS 1 r <- assigning value 1 to variable r
    - ASS f n <- assigning value of f to var n
    - LOG a <-outputting value of a (which is initially 0) <-0th fib value
    - ASS 1 b <-assigning value 1 to a variable b
    - JUM GRE n 1 7 16 <-if n is greater than 1 goto line 7(LOG B),else line 16(JUM GRE n 1 8 20)
    - LOG b <-output value b <-1st fib value
    - ADD b a <-add b and a; result in R
    - ASS R r <- store value of R in variable r
    - ASS b a <- assign value b to var a
    - ASS r b <- assign value of r to var b
    - SUB n 1 <-subtract 1 from n; result in R
    - ASS R n <-store value R in variable n
    - LOG r <- output value r; 2nd till n fib values
    - JUM GRE n 1 8 20 <- if n>1 goto line 8(ADD b a) else line 20 (‘’)
    - JUM EQL f 1 17 19 <- if f==1 go to line 17(ASS 1 r) else line 19
    - ASS 1 r <-assign 1 to variable r
    - LOG r <-output r; 1st fib value
    - ASS a a <-assign a to a; do nothing

- Turing complete:
    - A form of conditional repetition or conditional jump
        - JUM instruction
    - A way to read and write some form of storage
        - variables are available
    - All needed arithmetic sub blocks for computation