* Goal is to verify if a block chain is valid
* Program must only accept one argument…the name of the text file
  + Could error check this, but also may want to skip this to make it fast
* Output:
  + If valid: print out all of the addresses which have billcoins and how many
  + If invalid: print out exactly where the error is
* Program name: verifier.rb
* Repository must be named: D4
* Must use minitests for unit tests
* Must used rubocop with attached .rubocop.yml
* And must use SimpleCov
* Must use flamegraph to find execution “hot spots”
* Use measure command to determine total execution time
* Must answer:
  + What was most challenging about this deliverable?
  + What kind of edge cases and failure modes did you consider?
  + Using the flame graph, what methods were taking up the most CPU time?
  + What changes did you make based on the flame graph and timing?
* MUST HAVE TWO SCREENSHOTS OF FLAME GRAPH..ONE BEFORE MAKING PERFORMANCE ENHANCEMENTS, AND ONE AFTER MAKING PERFORMANCE ENHANCEMENTS
* ALSO INCLUDE THE GENERATED .HTML FILE IN YOUR REPOSITORY
* CAN USE ANY FILE FOR FLAME GRAPH
* 20 unit tests, statement coverage of at least 90%
* Time program with long.txt three times and indicate the mean and meadian amount of real time it took to execute
  + Measure-command in windows
* First get program to work, then refractor
* Tag the initial but non optimized commit with the tag INITITAL
  + Can then use git diff INITIAL to see the changes made
* Rubocop only on final version
* To add tag use:
  + Git tag -a INITITAL -m “INITIAL”
  + Git push origin INITIAL
* Billcoin
  + Block chains hold a list of transactions associated with a block
  + RULES:
    - Block 1 must come before block 2
    - Block 1 created after block 0(check times)
    - No transactions where someone gives away more coins than they own
    - HASHES SHOULD BE IN LOWERCASE
    - NO LEADING 0’S IN HASH
    - ONLY 4 CHARACTERS LONG AT MOST
    - Block number must be only incremented by 1 each time
    - USER ADDRESSES MAY ONLY BE 6 DECIMAL DIGITS CONSISTING OF THE NUMBER 0..9
      * LONGER THAN 6 INVALID
      * ANY NON-NUMBERIC CHARACTER INVALID
      * LESS THAN 6 DIGITS INVALID
    - Last transaction of a block must always be from SYSTEM
    - Every block must have at least this transaction
    - Block 0 should have only ONE transaction, the reward block, because at the beginning of the block chain, nobody has any billcoins to transfer
    - Timestamp must have a period
    - SHOULD NEVER BE THE SAME AS PREVIOUS OR MOVE BACKWARDS
      * NOTE: NOT FLOATING POINT, FIRST NUMBER IS LEFTMOST OF PERIOD, SECOND NUMBER RIGHTMOST OF PERIOD IN TIME STAMP
    - System>addr must pass 100 but not really giving 100, just verifying values
  + Hash function for bill coin
    - Takes the value for each characters UTF-8 value
    - ((x\*\*3000) + (x\*\*x) – (3\*\*x)) \* (7\*\*x)
    - Take the result of each character, and sum it up
    - Then take mod 65536 and display as up to 4 hex decimals
      * NOTE: leading 0’s in hex will not be displayed
    - ALL HEXIDECIMAL VALUES SHOULD BE LOWERCASE
    - Need to ensure next block was created after previous by hashing the previous block’s hash value as part of our data string
      * Add it to value before mod??
    - ALL BLOCKS HAVE A “REWARD” AS THE LAST TRANSACTION IN THEM??
  + ONE OPTIMIZATION:
    - Only calculate hash after verifying everything else checks out as hash will take a lot of time to compute
  + BLOCK ARRANGEMENT
    - Items in blocks separated by | characters
    - First block number starts at 0…incremented by 1 each time
      * Hash for first block just says 0 as no previous hash
    - Second group is the hash of the previous block
    - Third block is a sequence of transactions separated by :
      * In form FROM>TO
      * Addresses must be exactly 6 numeric characters
      * Value sending in parentheses
      * Last transaction is SYSTEM itself
        + Reward for creating a block, given to random user
      * Last transaction of a block must always be from SYSTEM
    - Fourth block
      * Time stamp of seconds since the epoch
      * A period
      * And the number of nanoseconds in that second when the block was created
    - 5TH BLOCK
      * A hash of the first 4 elements
      * Each element is the value between the pipes
      * String to hash can be generated via:
        + Convert to UTF-8 value
        + Perform calculation for each character in string
        + Sum all of values up
        + take mod
        + return resulting value as a string of the number in base 16(HEX)
        + see hash\_walkthrough.txt
      * HASH INCLUDES THE | PIPES, EXCEPT FOR THE LAST |
* Transactions valid on a per-block basis
* There may be a point midway through the block that an address may have negative balance, but by end must be >= 0
  + Need some way of keeping track of balance of each block..some sort of table
  + Also need set flags if some address less than 0 so that by end, if flag still set, then inaccurate
* MAY HAVE ISSUES WITH EXTRA CHARACTERS BETWEEN OR AT PIPES
  + AT EACH PIPE, ENSURE NEXT CHARACTER IS NOT A PIPE
* The SYSTEM>addr(100) does not give addr 100…
* Optimizations
  + Check to see if number of coins transferring is valid before checking addresses?
  + Check to see if period in time stamp before checking whole value
* Use SAMPLE.TXT FOR FLAMEGRPAH, LONG.TXT FOR TIMES