Ragaraja 1.0: The Genome Interpreter of Digital Organism Simulation Environment (DOSE)

Maurice HT Ling
Department of Mathematics and Statistics
South Dakota State University, USA
Department of Zoology
The University of Melbourne, Australia
mauriceling@acm.org

Abstract

This manuscript documents the implementation of Ragaraja interpreter version 1.0, the 3-character genetic code interpreter in Digital Organisms Simulation Environment (DOSE). These codes are licensed under Python Software Foundation License version 2.

1. Description

Christopher Langton (1986) had conceptualized that by casting chemical reactions, reactants, and products into computable operations, operands, and outputs respectively, it may be possible to simulate artificial life (organisms in the digital world, or digital organisms) as cellular automata "living" on artificial chemistries. This created the field of artificial life (ALife) and had been used in many different domains (Ward et al., 2011; Kim and Cho, 2006). Bersini (2009) argued that artificial life and theoretical biology shared many common grounds and presented genetic algorithm (GA) as an important model to bridge the two fields. GA was used to study microbial genetics and evolution (Harvey, 2011; Watson, 2012).

A recent Python implementation of GA, grounded on the biological hierarchy from genome to organisms to population (Lim et al., 2010), had been used as a basis to develop into a simulation framework for digital organisms, known as Digital Organisms Simulation Environment (DOSE) (Ling, 2012). The genetic material of DOSE organisms is an executable DNA based on a parameterless 3-character instruction code, known as Ragaraja, which Ling (2012) argued that such an instruction set mimics the naturally occurring DNA. This manuscript documents the implementation of Ragaraja interpreter version 1.0 which is licensed under Python Software Foundation License version 2.

At the core of Ragaraja interpreter is a 1-dimensional register machine (*file name: register_machine.py*). This register machine is implemented as a generic processor which does not contain the operations for each instruction. The individual operations have to be given to the interpreter routine as dictionary.

Each operation takes a tuple of 6 parameters – an array which represents the virtually limitless tape of a Turing machine, an array pointer to represent the read/write head on the tape, list of input for the instruction to use if needed, list of output to collect any output from the execution of the instruction, source tape to represent the string of instructions to execute, and the source pointer to represent the positional value of the instruction within the source tape. This implies that all needed input has to be given at the start of execution. Each operation will return a tuple of the same 6 parameters. Therefore, the execution of a set of instruction is nothing more than a loop around the instructions and feeding the outcome from the previously executed instruction as input to the next instruction to execute. An important feature of this interpreter is that the array (Turing tape) is implemented as a circular list; thus, allowing it to be "virtually limitless".

$$(array, apointer, input, output, source, spointer) \xrightarrow{instruction} (array', apointer', input', output', source', spointer')$$

As the register machine is made generic and execution of source is designed as a loop, the execution of each instruction can be self-contained (*file name: rajaraja.py*). The definition of each instruction for Rajaraga version 1.0 had been given in Ling (2012). As mentioned in Ling (2012), a subset of Ragaraja version 1.0, known as Nucleotide BF, had been defined as version 0.1. The instruction set for NucleotideBF consists of single alphabets and can be converted to Ragaraja instructions using *nBF_to_Ragaraja* function. In addition, *source_filter* function is provided to remove instructions that are not defined by the current version of Ragaraja from the source string.

2. Code Files

File name: register_machine.py

```
'''
One dimensional tape/register machine
Date created: 15th August 2012
Licence: Python Software Foundation License version 2

The machine consists of the following elements:
1. Array/Tape: A circular tape for operations to occur
2. Source: The program
3. Input List: A list of data given to the machine at initialization.
4. Output List: A list of output from the execution. This may also be
used as a secondary tape.

When the program terminates, all 4 elements are returned, and the
machine terminates itself.
'''

def interpret(source, functions,
             function_size=1, inputdata=[],
             array=None, size=30, max_instructions=1000):
    '''
    Interpreter loop.

    @param source: Instructions to execute.
    @type source: string
    @param functions: Dictionary of functions / operations.
    @param function_size: Length of each instruction. Default = 1
    @type function_size: integer
    @param inputdata: Any input data that the function may need.
    @type inputdata: list
    @param array: The endless tape in a Turing machine which is implemented
    as a circular list, making it virtually limitless.
    @type array: list
    @param size: Length of the type (array). Default = 30
    @type size: integer
    @param max_instructions: The maximum number of instructions to execute.
    Default = 1000
    @type max_instructions: integer
    '''
    spointer = 0
    apointer = 0
    output = list()
    if array == None: array = [0] * size
    if len(array) > size: array = array[0:size]
    if len(source) % function_size != 0:
        source = source + '!'*(function_size - \
                               len(source) % function_size)
       tokens = functions.keys()
       source = ''.join([x for x in source if x in tokens])
    instruction_count = 0
    while spointer < len(source):
        instruction_count = instruction_count + 1
        try:
            cmd = source[spointer:spointer+function_size]
            #print instruction_count, cmd
            (array, apointer, inputdata, output,
                source, spointer) = functions[cmd](array, apointer,
                                                   inputdata, output,
                                                   source, spointer)
        except KeyError:
            print ' '.join(['Unknown function: ', cmd,
                            'at source position', str(spointer)])
        if apointer > size - 1: apointer = apointer - size
        if apointer < 0: apointer = size + apointer
        spointer = spointer + function_size
        if instruction_count > max_instructions:
            return (array, apointer, inputdata, output, source, spointer)
    return (array, apointer, inputdata, output, source, spointer)
```

The following functions are defined in *lc_bf.py* and used in *ragaraja.py*:

```
def increment(array, apointer, inputdata, output, source, spointer):
    '''Increase value of cell by 1. Equivalent to "+" in Brainfuck.'''
    array[apointer] = array[apointer] + 1
    return (array, apointer, inputdata, output, source, spointer)

def decrement(array, apointer, inputdata, output, source, spointer):
    '''Decrease value of cell by 1. Equivalent to "-" in Brainfuck.'''
    array[apointer] = array[apointer] - 1
    return (array, apointer, inputdata, output, source, spointer)

def forward(array, apointer, inputdata, output, source, spointer):
    '''Move forward by one cell on tape. Equivalent to ">" in Brainfuck.'''
    return (array, apointer + 1, inputdata, output, source, spointer)

def backward(array, apointer, inputdata, output, source, spointer):
    '''Move backward by one cell on tape. Equivalent to "<" in Brainfuck.'''
    return (array, apointer - 1, inputdata, output, source, spointer)

def call_out(array, apointer, inputdata, output, source, spointer):
    '''Output current tape cell value and append to the end of the
    output list. Equivalent to "." in Brainfuck.'''
    output.append(array[apointer])
    return (array, apointer, inputdata, output, source, spointer)

def accept_predefined(array, apointer, inputdata, output, source, spointer):
    '''Writes the first value of the input list into the current cell and remove
    the value from the input list. If input list is empty, "0" will be written'''
    if len(inputdata) > 0: array[apointer] = inputdata.pop(0)
    else: array[apointer] = 0
    return (array, apointer, inputdata, output, source, spointer)
```

File name: ragaraja.py

```
'''
Ragaraja Interpreter
Date created: 16th August 2012
Licence: Python Software Foundation License version 2

Ragaraja is a derivative and massive extension of Brainfuck. This work is
influenced by a large number of Brainfuck derivatives, other esoteric programming
languages, and even assembly languages. Probably the most critical difference
between Ragaraja and other Brainfuck derivatives is the large
number of commands / instructions - 1000 possible commands / instructions, inspired
by Nandi (follower of Lord Shiva) who was supposed to be the first author of Kama
Sutra and wrote it in 1000 chapters.

Etymology: Ragaraja is the name of a Mahayana Buddhist deity from Esoteric
traditions. The Chinese calls him Ai Ran Ming Wang. Ragaraja is one of the Wisdom
Kings (a group of Bodhisattvas) and represents the state at which sexual excitement
or agitation can be channeled towards enlightenment and passionate love can become
compassion for all living things. Hence, I name this
Compilation / derivative / extension of Brainfuck in 1000 commands / instructions /
opcode to signify the epitome, a channeling of raw urge to the love and compassion
for and towards every being. May really be viewed as Brainfuck attaining
enlightenment or Nirvana. Whoever that can remember all 1000 commands and use it,
really deserves an award.

The interpreter environment consists of the following elements:

1. Array/Tape: A circular tape initialized with 30 thousand cells each with zero.
This can be visualized as a 30,000 cell register machine. The number of cells can
increase or decrease during runtime.
2. Source: The program
3. Input List: A list of data given to the execution environment at initialization.
4. Output List: A list of output from the execution. This may also be used as a
secondary tape.

When the program terminates, all 4 elements are returned, and the
interpreter terminates itself.

Ref: http://esolangs.org/wiki/Ragaraja
'''
import random
import math
from samplestatistics import SingleSample
from lc_bf import increment, decrement
from lc_bf import forward, backward
from lc_bf import call_out, accept_predefined

register = [0]*99

def instruction_padding(inst):
       inst = str(inst)
       if len(inst) == 1: return '00' + inst
       if len(inst) == 2: return '0' + inst
       if len(inst) == 3: return inst

def loop_start(array, apointer, inputdata, output, source, spointer):
    '''
    Start loop. Will only enter loop if current cell is more than "0". If
    current cell is "0" or less, it will go to the end of the loop
    (command 015). if the loop is not closed, it will go to the end of the
    source.
    '''
    if array[apointer] > 0:
        return (array, apointer, inputdata, output, source, spointer)
    else:
        count = 1
        try:
            while count > 0:
                spointer = spointer + 3
                if source[spointer:spointer+3] == '015': count = count - 1
                if source[spointer:spointer+3] == '014': count = count + 1
            return (array, apointer, inputdata, output, source, spointer - 3)
        except IndexError:
            return (array, apointer, inputdata, output, source,
                    len(source) - 1)

def loop_end(array, apointer, inputdata, output, source, spointer):
    '''
    End loop. However, it is possible to have an end loop operator
    (command 015) without a preceding start loop operator (command 014).
    In this case, the end loop operator (command 015) will be ignored and
    execution continues.
    '''
    temp = spointer
    if array[apointer] < 1:
        return (array, apointer, inputdata, output, source, spointer)
    else:
        count = 1
        try:
            while count > 0:
                spointer = spointer - 3
                if source[spointer:spointer+3] == '015': count = count + 1
                if source[spointer:spointer+3] == '014': count = count - 1
        except IndexError:
            spointer = temp
    return (array, apointer, inputdata, output, source, spointer)

def tape_move(array, apointer, inputdata, output, source, spointer):
    '''
    Moving tape pointer for more than one increment or decrement.

    Instructions handled:
    001: Move forward by 5 cells on tape. Equivalent to 5 times of
    "000".
    002: Move forward by 10 cells on tape. Equivalent to 10 times of
    "000".
    003: Move forward by NxN cells on tape where N is the value of
    the current cell. If N is a decimal, it will move forward by the
    floor of NxN. For example, if N is 4.2, this operation will tape
    pointer forward by 17 cells. As NxN is always a positive number,
    it does not matter if the value of the current cell is positive
    or negative.
    005: Move backward by 5 cells on tape. Equivalent to 5 times of
    "004".
    006: Move backward by 10 cells on tape. Equivalent to 10 times of
    "004".
    007: Move backward by NxN cells on tape where N is the value of the
    current cell. If N is a decimal, it will move backward by the floor
    of NxN. For example, if N is 4.2, this operation will tape pointer
    backward by 17 cells. As NxN is always a positive number, it does
    not matter if the value of the current cell is positive or negative.
    043: Move the tape cell pointer to the first cell.
    044: Move the tape cell pointer to the last cell.
    045: Move the tape cell pointer to the location determined by the
    last value of the output list. If the last value of the output list
    is more than the length of the tape, it will take the modulus of
    the length of the tape. For example, the last value of the output
    list is 5, the tape cell pointer will point to the 5th cell on the tape.
    061: Move forward by the number of cells signified by the current cell.
    062: Move backward by the number of cells signified by the current cell.
    140: Move tape pointer to the centre of the tape. If the tape has
    odd number cells, it will move to the lower cell. For example, this
    instruction will move the tape pointer to the 500th cell of a
    1000-cell tape, or 142nd of a 285-cell tape.
    141: Move tape pointer to 1/4 the of the tape. If the tape has odd
    number cells, it will move to the lower cell. For example, this
    instruction will move the tape pointer to the 250th cell of a
    1000-cell tape, or 71st of a 285-cell tape.
    142: Move tape pointer to 3/4 the of the tape. If the tape has odd
    number cells, it will move to the lower cell. For example, this
    instruction will move the tape pointer to the 750th cell of a
    1000-cell tape, or 213rd of a 285-cell tape.
    143: Move tape pointer to the position as the integer value in the
    current cell. If the value of the cell is larger than the length of
    the tape, it will move to the modulus of the integer value in the
    current cell.
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '001': apointer = apointer + 5
    if cmd == '002': apointer = apointer + 10
    if cmd == '003':
        move = int(float(array[apointer]) * float(array[apointer]))
        apointer = apointer + move
    if cmd == '005': apointer = apointer - 5
    if cmd == '006': apointer = apointer - 10
    if cmd == '007':
        move = int(float(array[apointer]) * float(array[apointer]))
        apointer = (apointer - move) % (len(array) - 1)
    if cmd == '043': apointer = 0
    if cmd == '044': apointer = len(array) - 1
    if cmd == '045': apointer = int(output[-1]) % (len(array) - 1)
    if cmd == '061': apointer = apointer + int(array[apointer])
    if cmd == '062': apointer = apointer - int(array[apointer])
    if cmd == '140': apointer = int((len(array) - 1) * 0.5)
    if cmd == '141': apointer = int((len(array) - 1) * 0.25)
    if cmd == '142': apointer = int((len(array) - 1) * 0.75)
    if cmd == '143': apointer = int(array[apointer]) % (len(array) - 1)
    return (array, apointer, inputdata, output, source, spointer)

def accumulations(array, apointer, inputdata, output, source, spointer):
    '''
    Accumulate the tape cell by more than one increment or decrement.

    Instructions handled:
    009: Increase value of cell by 5. Equivalent to 5 times of "008".
    010: Increase value of cell by 10. Equivalent to 10 times of "008".
    012: Decrease value of cell by 5. Equivalent to 5 times of "011".
    013: Decrease value of cell by 10. Equivalent to 10 times of "011".
    032: Double current tape cell value.
    033: Half current tape cell value.
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '009': array[apointer] = array[apointer] + 5
    if cmd == '010': array[apointer] = array[apointer] + 10
    if cmd == '012': array[apointer] = array[apointer] - 5
    if cmd == '013': array[apointer] = array[apointer] - 10
    if cmd == '032': array[apointer] = 2 * array[apointer]
    if cmd == '033': array[apointer] = 0.5 * array[apointer]
    return (array, apointer, inputdata, output, source, spointer)

def nBF_random_op(array, apointer, inputdata, output, source, spointer):
    '''
    NucleotideBF (nBF) random operations - to simulate ambiguous DNA bases.

    Instructions handled:
    050: Randomly execute "008" (increment by 1) or "000" (move forward
    by 1). Equivalent to "R" in NucleotideBF (nBF).
    051: Randomly execute "011" (decrement by 1) or "004" (move backward
    by 1). Equivalent to "Y" in NucleotideBF (nBF).
    052: Randomly execute "000" (move forward by 1) or "004" (move backward
    by 1). Equivalent to "S" in NucleotideBF (nBF).
    053: Randomly execute "008" (increment by 1) or "011" (decrement by 1).
    Equivalent to "W" in NucleotideBF (nBF).
    054: Randomly execute "000" (move forward by 1) or "011" (decrement
    by 1). Equivalent to "K" in NucleotideBF (nBF).
    055: Randomly execute "004" (move backward by 1) or "008" (increment
    by 1). Equivalent to "M" in NucleotideBF (nBF).
    056: Randomly execute "000" (move forward by 1) or "004" (move backward
    by 1) or "011" (decrement by 1). Equivalent to "B" in NucleotideBF (nBF).
    057: Randomly execute "000" (move forward by 1) or "008" (increment by 1)
    or "011" (decrement by 1). Equivalent to "D" in NucleotideBF (nBF).
    058: Randomly execute "004" (move backward by 1) or "008" (Increment
    by 1) or "011" (decrement by 1). Equivalent to "H" in NucleotideBF (nBF).
    059: Randomly execute "000" (move forward by 1) or "004" (move backward
    by 1) or "008" (increment by 1). Equivalent to "V" in NucleotideBF (nBF).
    060: Randomly execute "000" (move forward by 1) or "004" (move backward
    by 1) or "008" (increment by 1) or "011" (decrement by 1). Equivalent
    to "N" in NucleotideBF (nBF)
    '''
    cmd = source[spointer:spointer+3]
    r = random.random()
    if cmd == '050' and r < 0.5:
        return increment(array, apointer, inputdata, output, source, spointer)
    elif cmd == '050' and r >= 0.5:
        if (apointer + 1) == len(array):
            return (array, 0, inputdata, output, source, spointer)
        else:
            return forward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '051' and r < 0.5:
        return decrement(array, apointer, inputdata, output, source, spointer)
    elif cmd == '051' and r >= 0.5:
        if apointer == 0:
            return (array, len(array) - 1, inputdata, output, source, spointer)
        else:
            return backward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '052' and r < 0.5:
        if (apointer + 1) == len(array):
            return (array, 0, inputdata, output, source, spointer)
        else:
            return forward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '052' and r >= 0.5:
        if apointer == 0:
            return (array, len(array) - 1, inputdata, output, source, spointer)
        else:
            return backward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '053' and r < 0.5:
        return increment(array, apointer, inputdata, output, source, spointer)
    elif cmd == '053' and r >= 0.5:
        return decrement(array, apointer, inputdata, output, source, spointer)
    elif cmd == '054' and r < 0.5:
        return decrement(array, apointer, inputdata, output, source, spointer)
    elif cmd == '054' and r >= 0.5:
        if (apointer + 1) == len(array):
            return (array, 0, inputdata, output, source, spointer)
        else:
            return forward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '055' and r < 0.5:
        return increment(array, apointer, inputdata, output, source, spointer)
    elif cmd == '055' and r >= 0.5:
        if apointer == 0:
            return (array, len(array) - 1, inputdata, output, source, spointer)
        else:
            return backward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '056' and r < 0.33:
        if (apointer + 1) == len(array):
            return (array, 0, inputdata, output, source, spointer)
        else:
            return forward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '056' and r >= 0.33 and r < 0.67:
        return decrement(array, apointer, inputdata, output, source, spointer)
    elif cmd == '056' and r >= 0.67:
        if apointer == 0:
            return (array, len(array) - 1, inputdata, output, source, spointer)
        else:
            return backward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '057' and r < 0.33:
        return increment(array, apointer, inputdata, output, source, spointer)
    elif cmd == '057' and r >= 0.33 and r < 0.67:
        return decrement(array, apointer, inputdata, output, source, spointer)
    elif cmd == '057' and r >= 0.67:
        if (apointer + 1) == len(array):
            return (array, 0, inputdata, output, source, spointer)
        else:
            return forward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '058' and r < 0.33:
        return increment(array, apointer, inputdata, output, source, spointer)
    elif cmd == '058' and r >= 0.33 and r < 0.67:
        return decrement(array, apointer, inputdata, output, source, spointer)
    elif cmd == '058' and r >= 0.67:
        if apointer == 0:
            return (array, len(array) - 1, inputdata, output, source, spointer)
        else:
            return backward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '059' and r < 0.33:
        return increment(array, apointer, inputdata, output, source, spointer)
    elif cmd == '059' and r >= 0.33 and r < 0.67:
        if (apointer + 1) == len(array):
            return (array, 0, inputdata, output, source, spointer)
        else:
            return forward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '059' and r >= 0.67:
        if apointer == 0:
            return (array, len(array) - 1, inputdata, output, source, spointer)
        else:
            return backward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '060' and r < 0.25:
        return increment(array, apointer, inputdata, output, source, spointer)
    elif cmd == '060' and r >= 0.25 and r < 0.5:
        return decrement(array, apointer, inputdata, output, source, spointer)
    elif cmd == '060' and r >= 0.5 and r < 0.75:
        if (apointer + 1) == len(array):
            return (array, 0, inputdata, output, source, spointer)
        else:
            return forward(array, apointer, inputdata, output, source, spointer)
    elif cmd == '060' and r >= 0.75:
        if apointer == 0:
            return (array, len(array) - 1, inputdata, output, source, spointer)
        else:
            return backward(array, apointer, inputdata, output, source, spointer)

def tape_size(array, apointer, inputdata, output, source, spointer):
    '''
    Change the length of the tape during runtime.

    Instructions handled:
    016: Add one cell to the end of the tape.
    017: Add 10 cells to the end of the tape.
    018: Remove one cell from the end of the tape. If original tape pointer
    is at the last cell before removal operation, the tape pointer will point
    to the last cell after removal.
    019: Remove 10 cells from the end of the tape. If original tape pointer
    is at the last cell before removal operation, the tape pointer will point
    to the last cell after removal.
    034: Insert a cell after the current tape cell. For example, if current
    tape cell is 35, a cell initialized to zero will be added as cell 36. As
    a result, the tape is 1 cell longer.
    035: Delete the current cell. As a result, the tape is 1 cell shorter.
    036: Delete the current and append to the end of the output list. As a
    result, the tape is 1 cell shorter.
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '016': array = array + [0]
    if cmd == '017': array = array + [0]*10
    if cmd == '018': array = array[:-1]
    if cmd == '019': array = array[:-10]
    if cmd == '034': array.insert(apointer + 1, 0)
    if cmd == '035': array.pop(apointer)
    if cmd == '036': output.append(array.pop(apointer))
    if apointer >= len(array): apointer = len(array) - 1
    return (array, apointer, inputdata, output, source, spointer)

def source_move(array, apointer, inputdata, output, source, spointer):
    '''
    Moving the source without execution.

    Instructions handled:
    023: Move source pointer forward by one instruction without execution
    if the source pointer does not point beyond the length of the source
    after the move, otherwise, does not move the source pointer.
    024: Move source pointer forward by 5 instruction without execution
    if the source pointer does not point beyond the length of the source
    after the move, otherwise, does not move the source pointer.
    025: Move source pointer forward by 10 instruction without execution
    if the source pointer does not point beyond the length of the source
    after the move, otherwise, does not move the source pointer.
    026: Move source pointer backward by one instruction without execution
    if the source pointer does not point beyond the length of the source
    after the move, otherwise, does not move the source pointer.
    027: Move source pointer backward by 5 instruction without execution
    if the source pointer does not point beyond the length of the source
    after the move, otherwise, does not move the source pointer.
    028: Move source pointer backward by 10 instruction without execution
    if the source pointer does not point beyond the length of the source
    after the move, otherwise, does not move the source pointer.
    082: Skip next instruction if current cell is "0". Equivalent to "/"
    in [[Minimal]]. However, this operation will only execute if there is
    at least 1 more instruction from the current instruction.
    083: Skip the number of instructions equivalent to the absolute integer
    value of the current cell if the source pointer does not point beyond
    the length of the source after the move, otherwise, does not move the
    source pointer. For example, if current cell is "5.6" or "5", the next
    5 instructions will be skipped.
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '023' and (spointer + 3) < len(source):
        spointer = spointer + 3
    if cmd == '024' and (spointer + 15) < len(source):
        spointer = spointer + 15
    if cmd == '025' and (spointer + 30) < len(source):
        spointer = spointer + 30
    if cmd == '026' and (spointer - 3) >= 0: spointer = spointer - 3
    if cmd == '027' and (spointer - 15) >= 0: spointer = spointer - 15
    if cmd == '028' and (spointer - 30) >= 0: spointer = spointer - 30
    if cmd == '082' and array[apointer] == 0 and \
    (spointer + 3) <= len(source):
        spointer = spointer + 3
    if cmd == '083' and \
    (spointer + (3 * abs(int(array[apointer])))) < len(source):
        spointer = spointer + (3 * abs(int(array[apointer])))
    return (array, apointer, inputdata, output, source, spointer)
```

```
def set_tape_value(array, apointer, inputdata, output, source, spointer):
    '''
    Set values into tape cell by over-writing the original value.

    Instructions handled:
    084: Set current tape cell to "0".
    085: Set current tape cell to "-1".
    086: Set current tape cell to "1".
    097: Set the value of the current cell to pi (3.14159265358979323846)
    098: Set the value of the current cell to e (2.718281828459045)
    187: Set all the cell values in the cells after the current cell to
    "0" (current cell is not affected).
    188: Set all the cell values in the cells before the current cell to
    "0" (current cell is not affected).
    189: Set all values in tape to "0".
    190: Set all the cell values in the tape to the value of current cell.
    191: Set all the cell values in the tape to the tape position of current cell.
    192: Set all the cell values in the cells after the current cell to
    the value of current cell.
    193: Set all the cell values in the cells before the current cell to
    the value of current cell.
    194: Set all the cell values in the cells after the current cell to
    the tape position of current cell.
    195: Set all the cell values in the cells before the current cell to
    the tape position of current cell.
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '084': array[apointer] = 0
    if cmd == '085': array[apointer] = -1
    if cmd == '086': array[apointer] = 1
    if cmd == '097': array[apointer] = constants.PI
    if cmd == '098': array[apointer] = math.e
    if cmd == '187':
        array = array[0:apointer+1] + [0 for x in array[apointer+1:]]
    if cmd == '188':
        array = [0 for x in array[:apointer]] + array[apointer:]
    if cmd == '189': array = [0] * len(array)
    if cmd == '190': array = [array[apointer]] * len(array)
    if cmd == '191': array = [apointer] * len(array)
    if cmd == '192':
        array = array[0:apointer+1] + [array[apointer]
                                       for x in array[apointer+1:]]
    if cmd == '193':
        array = [array[apointer] for x in array[:apointer]] + array[apointer:]
    if cmd == '194':
        array = array[0:apointer+1] + [apointer for x in array[apointer+1:]]
    if cmd == '195':
        array = [apointer for x in array[:apointer]] + array[apointer:]
    return (array, apointer, inputdata, output, source, spointer)

def mathematics(array, apointer, inputdata, output, source, spointer):
    '''
    Performs mathematical and arithmetical operations.

    Instructions handled:
    065: Add the value of the current cell (n) and (n+1)th cell, and
    store the value in the current cell. Array[n] = Array[n] + Array[n+1]
    066: Add the value of the current cell (n) and first of the input
    list, and store the value in the current cell.
    067: Add the value of the current cell (n) and last of the input
    list, and store the value in the current cell.
    068: Subtract the value of the current cell (n) from (n+1)th
    cell, and store the value in the current cell.
    Array[n] = Array[n+1] - Array[n]
    069: Subtract the value of the current cell (n) from the first
    of the input list, and store the value in the current cell.
    Array[n] = InputList[0] - Array[n]
```

```
070: Subtract the value of the current cell (n) from the last
of the input list, and store the value in the current cell.
Array[n] = InputList[-1] - Array[n]
071: Multiply the value of the current cell (n) and (n+1)th
cell, and store the value in the current cell.
Array[n] = Array[n+1] * Array[n]
072: Multiply the value of the current cell (n) and first of
the input list, and store the value in the current cell.
073: Multiply the value of the current cell (n) and last of
the input list, and store the value in the current cell.
074: Divide the value of the current cell (n) from (n+1)th
cell, and store the value in the current cell.
Array[n] = Array[n+1] / Array[n]
075: Divide the value of the current cell (n) from the first
of the input list, and store the value in the current cell.
Array[n] = InputList[0] / Array[n]
076: Divide the value of the current cell (n) from the last
of the input list, and store the value in the current cell.
Array[n] = InputList[-1] - Array[n]
077: Modulus (remainder after division) the value of the
current cell (n) from (n+1)th cell, and store the value in
the current cell. Array[n] = Array[n+1] % Array[n]
078: Modulus (remainder after division) the value of the
current cell (n) from the first of the input list, and store
the value in the current cell. Array[n] = InputList[0] % Array[n]
079: Modulus (remainder after division) the value of the
current cell (n) from the last of the input list, and store
the value in the current cell. Array[n] = InputList[-1] % Array[n]
080: Floor the value of the current cell. For example, if
the value of the current cell is 6.7, it will becomes 6.
087: Negate the value of the current cell. Positive value will be
negative. Negative value will be positive. Equivalent to "_" in L00P
088: Calculate the sine of the value of the current cell (measured
in radians) and replace. Equivalent to "s" in Grin.
Array[n] = sine(Array[n])
089: Calculate the cosine of the value of the current cell
(measured in radians) and replace. Equivalent to "c" in Grin.
Array[n] = cosine(Array[n])
090: Calculate the tangent of the value of the current cell
(measured in radians) and replace. Equivalent to "t" in Grin.
Array[n] = tangent(Array[n])
091: Calculate the arc sine of the value of the current cell
(measured in radians) and replace. Equivalent to "S" in Grin.
Array[n] = arcsine(Array[n])
092: Calculate the arc cosine of the value of the current cell
(measured in radians) and replace. Equivalent to "C" in Grin.
Array[n] = arccosine(Array[n])
093: Calculate the arc tangent of the value of the current cell
(measured in radians) and replace. Equivalent to "T" in Grin.
Array[n] = arctangent(Array[n])
094: Calculate the reciprocal of the value of the current cell
(measured in radians) and replace. Equivalent to "1" in Grin.
Array[n] = 1/Array[n]
095: Calculate the square root of the value of the current cell
(measured in radians) and replace. Equivalent to "q" in Grin.
Array[n] = sqrt(Array[n])
096: Calculate the natural logarithm of the value of the current
cell (measured in radians) and replace. Equivalent to "l" in Grin.
Array[n] = ln(Array[n])
099: Calculate the hyperbolic sine of the value of the current
cell (measured in radians) and replace. Array[n] = sinh(Array[n])
100: Calculate the hyperbolic cosine of the value of the current
cell (measured in radians) and replace. Array[n] = cosh(Array[n])
101: Calculate the hyperbolic tangent of the value of the current
cell (measured in radians) and replace. Array[n] = tanh(Array[n])
102: Calculate the hyperbolic arc sine of the value of the current
cell (measured in radians) and replace. Array[n] = arcsinh(Array[n])
```

103: Calculate the hyperbolic arc cosine of the value of the current cell (measured in radians) and replace. Array[n] = arccosh(Array[n])

104: Calculate the hyperbolic arc tangent of the value of the current cell (measured in radians) and replace. Array[n] = arctanh(Array[n])

105: Convert the value of the current cell (measured in radians) to degrees and replace.

106: Convert the value of the current cell (measured in degrees) to radians and replace.

107: Raise the value of the current cell (n) to e, and store the value in the current cell. Array[n] = Array[n]^e

108: Raise e to the value of the current cell (n), and store the value in the current cell. Array[n] = e^Array[n]

109: Raise 10 to the value of the current cell (n), and store the value in the current cell. Array[n] = 10^Array[n]

110: Raise the value of the current cell (n) to the value of (n+1)th cell, and store the value in the current cell. Array[n] = Array[n]^Array[n+1]

111: Calculate the n-th root of the value of the current cell (n) where n is the value of (n+1)th cell, and store the value in the current cell. Array[n] = Array[n]^(1/Array[n+1])

112: Calculate the error function of the value of the current cell and replace. Array[n] = erf(Array[n])

113: Calculate the complementary error function of the value of the current cell and replace. Array[n] = erfc(Array[n])

114: Calculate the factorial of the integer value of the current cell (if the integer value is positive) and replace. Array[n] = factorial(Array[n])

115: Calculate the factorial of the absolute integer value of the current cell and replace. Array[n] = factorial(abs(Array[n]))

116: Calculate the Euclidean distance (hypotenuse) value of the current cell (n) to the value of (n+1)th cell, and store the value in the current cell. Array[n] = sqrt(Array[n]*Array[n] + Array[n+1]*Array[n+1])

117: Calculate the logarithm value of the current cell (n) to the base of the value of (n+1)th cell, and store the value in the current cell. Array[n] = log(Array[n], base=Array[n+1])

144: Divide current cell value by 10.

145: Multiply current cell value by 10.

146: Add all cell values from (n+1)th cell to the end of the tape and store result in current cell (n). Array[n] = sum(Array[n+1:])

147: Add all cell values from n-th cell to the end of the tape and store result in current cell (n). Array[n] = sum(Array[n:])

148: Add all cell values from first cell to the cell before n-th cell and store result in current cell (n). Array[n] = sum(Array[0:n])

149: Add all cell values from first cell to n-th cell (inclusive) and store result in current cell (n). Array[n] = sum(Array[0:n+1])

150: Add all cell values in the tape and store result in current cell (n). Array[n] = sum(Array[:])

151: Average all cell values from (n+1)th cell to the end of the tape and store result in current cell (n). Array[n] = average(Array[n+1:])

152: Average all cell values from n-th cell to the end of the tape and store result in current cell (n). Array[n] = average(Array[n:])

153: Average all cell values from first cell to the cell before n-th cell and store result in current cell (n). Array[n] = average(Array[0:n])

154: Average all cell values from first cell to n-th cell (inclusive) and store result in current cell (n). Array[n] = average(Array[0:n+1])

155: Half every cell value in tape.

156: Double every cell value in tape.

157: Divide every cell value in tape by 10.

158: Multiply every cell value in tape by 10.

159: Divide every cell value in tape by 100.

160: Multiply every cell value in tape by 100.

165: Multiply every cell value in tape by -1.

166: Square all the cell values in the cells after the current cell (current cell value is not affected).

167: Square all the cell values in the cells before the current cell

```
(current cell value is not affected).
168: Square every cell value in tape.
169: Square root every cell value in tape.
170: Square root all the cell values in the cells after the current
cell (current cell value is not affected).
171: Square root all the cell values in the cells before the current
cell (current cell value is not affected).
196: Set the value of the current cell to the standard deviation of
the values in the tape.
197: Set the value of the current cell to the geometric mean of the
values in the tape.
198: Set the value of the current cell to the harmonic mean of the
values in the tape.
'''
cmd = source[spointer:spointer+3]
if cmd == '065':
    if (apointer + 1) < len(array):
        array[apointer] = array[apointer] + array[apointer+1]
    else:
        array[apointer] = array[apointer] + array[0]
if cmd == '066' and len(inputdata) > 0:
    array[apointer] = array[apointer] + inputdata[0]
if cmd == '067' and len(inputdata) > 0:
    array[apointer] = array[apointer] + inputdata[-1]
if cmd == '068':
    if (apointer + 1) < len(array):
        array[apointer] = array[apointer+1] - array[apointer]
    else:
        array[apointer] = array[0] - array[apointer]
if cmd == '069' and len(inputdata) > 0:
    array[apointer] = inputdata[0] - array[apointer]
if cmd == '070' and len(inputdata) > 0:
    array[apointer] = inputdata[-1] - array[apointer]
if cmd == '071':
    if (apointer + 1) < len(array):
        array[apointer] = array[apointer+1] * array[apointer]
    else:
        array[apointer] = array[0] * array[apointer]
if cmd == '072' and len(inputdata) > 0:
    array[apointer] = inputdata[0] * array[apointer]
if cmd == '073' and len(inputdata) > 0:
    array[apointer] = inputdata[-1] * array[apointer]
if cmd == '074':
    if (apointer + 1) < len(array):
        array[apointer] = float(array[apointer+1]) / array[apointer]
    else:
        array[apointer] = array[0] / array[apointer]
if cmd == '075' and len(inputdata) > 0:
    array[apointer] = float(inputdata[0]) / array[apointer]
if cmd == '076' and len(inputdata) > 0:
    array[apointer] = float(inputdata[-1]) / array[apointer]
if cmd == '077':
    if (apointer + 1) < len(array):
        array[apointer] = array[apointer+1] % array[apointer]
    else:
        array[apointer] = array[0] % array[apointer]
if cmd == '078' and len(inputdata) > 0:
    array[apointer] = inputdata[0] % array[apointer]
if cmd == '079' and len(inputdata) > 0:
    array[apointer] = inputdata[-1] % array[apointer]
if cmd == '080': array[apointer] = int(array[apointer])
if cmd == '087': array[apointer] = -1 * array[apointer]
if cmd == '088': array[apointer] = math.sin(array[apointer])
if cmd == '089': array[apointer] = math.cos(array[apointer])
if cmd == '090': array[apointer] = math.tan(array[apointer])
if cmd == '091': array[apointer] = math.asin(array[apointer])
if cmd == '092': array[apointer] = math.acos(array[apointer])
if cmd == '093': array[apointer] = math.atan(array[apointer])
if cmd == '094': array[apointer] = 1 / array[apointer]
if cmd == '095': array[apointer] = math.sqrt(array[apointer])
if cmd == '096': array[apointer] = math.log(array[apointer], math.e)
if cmd == '099': array[apointer] = math.sinh(array[apointer])
if cmd == '100': array[apointer] = math.cosh(array[apointer])
if cmd == '101': array[apointer] = math.tanh(array[apointer])
if cmd == '102': array[apointer] = math.asinh(array[apointer])
if cmd == '103': array[apointer] = math.acosh(array[apointer])
if cmd == '104': array[apointer] = math.atanh(array[apointer])
if cmd == '105': array[apointer] = math.degrees(array[apointer])
if cmd == '106': array[apointer] = math.radians(array[apointer])
if cmd == '107': array[apointer] = array[apointer] ** math.e
if cmd == '108': array[apointer] = math.e ** array[apointer]
if cmd == '109': array[apointer] = 10 ** array[apointer]
if cmd == '110':
    if (apointer + 1) < len(array):
        array[apointer] = array[apointer] ** array[apointer+1]
    else:
        array[apointer] = array[apointer] ** array[0]
if cmd == '111':
    if (apointer + 1) < len(array):
        array[apointer] = array[apointer] ** (1 / array[apointer+1])
    else:
        array[apointer] = array[apointer] ** (1 / array[0])
if cmd == '112': array[apointer] = math.erf(array[apointer])
if cmd == '113': array[apointer] = math.erfc(array[apointer])
if cmd == '114' and array[apointer] >= 0:
    array[apointer] = math.factorial(int(array[apointer]))
if cmd == '115':
    array[apointer] = math.factorial(abs(int(array[apointer])))
if cmd == '116':
    if (apointer + 1) < len(array):
        array[apointer] = math.hypot(array[apointer], array[apointer+1])
    else:
        array[apointer] = math.hypot(array[apointer], array[0])
if cmd == '117':
    if (apointer + 1) < len(array):
        array[apointer] = math.log(array[apointer], array[apointer+1])
    else:
        array[apointer] = math.log(array[apointer], array[0])
if cmd == '144': array[apointer] = 0.1 * (array[apointer])
if cmd == '145': array[apointer] = 10 * (array[apointer])
if cmd == '146': array[apointer] = sum(array[apointer+1:])
if cmd == '147': array[apointer] = sum(array[apointer:])
if cmd == '148': array[apointer] = sum(array[0:apointer])
if cmd == '149': array[apointer] = sum(array[0:apointer+1])
if cmd == '150': array[apointer] = sum(array)
if cmd == '151':
    temp = array[apointer+1:]
    array[apointer] = sum(temp) / float(len(temp))
if cmd == '152':
    temp = array[apointer:]
    array[apointer] = sum(temp) / float(len(temp))
if cmd == '153':
    temp = array[0:apointer]
    array[apointer] = sum(temp) / float(len(temp))
if cmd == '154':
    temp = array[0:apointer+1]
    array[apointer] = sum(temp) / float(len(temp))
if cmd == '155': array = [0.5 * x for x in array]
if cmd == '156': array = [2 * x for x in array]
if cmd == '157': array = [0.1 * x for x in array]
if cmd == '158': array = [10 * x for x in array]
if cmd == '159': array = [0.01 * x for x in array]
if cmd == '160': array = [100 * x for x in array]
if cmd == '165': array = [-1 * x for x in array]
```

```
    if cmd == '166':
        array = array[0:apointer+1] + [x * x for x in array[apointer+1:]]
    if cmd == '167':
        array = [x * x for x in array[:apointer]] + array[apointer:]
    if cmd == '168': array = [x * x for x in array]
    if cmd == '169': array = [math.sqrt(x) for x in array]
    if cmd == '170':
        array = array[0:apointer+1] + [math.sqrt(x)
                                       for x in array[apointer+1:]]
    if cmd == '171':
        array = [math.sqrt(x) for x in array[:apointer]] + array[apointer:]
    if cmd == '196':
        variance = SingleSample(array).variance()
        array[apointer] = math.sqrt(variance)
    if cmd == '197': array[apointer] = SingleSample(array).geometricMean()
    if cmd == '198': array[apointer] = SingleSample(array).harmonicMean()
    return (array, apointer, inputdata, output, source, spointer)

def output_IO(array, apointer, inputdata, output, source, spointer):
    '''
    Using output list as output storage or secondary tape, write and
    accept values from output list.

    Instructions handled:
    021: Output current tape cell location and append to the end of
    the output list.
    022: Output current source location and append to the end of the output list.
    037: Replace the current tape cell value with the last value of
    the output list, and delete the last value from the output list.
    038: Replace the current tape cell value with the last value of
    the output list, without deleting the last value from the output list.
    039: Replace the current tape cell value with the first value of
    the output list, and delete the first value from the output list.
    040: Replace the current tape cell value with the first value of
    the output list, without deleting the first value from the output list.
    041: Remove first value from the output list.
    042: Remove last value from the output list.
    172: Append all the cell values in the cells after the current cell
    to the end of output list (current cell is not appended to output list).
    173: Append all the cell values in the cells after the current cell
    to the end of output list (current cell is not appended to output
    list), then set the values of all the cells after the current cell
    to "0" (current cell value is not affected).
    174: Append all the cell values in the cells to the end of output list.
    175: Append all the cell values in the cells to the end of output list
    and set the tape to "0".
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '021': output.append(apointer)
    if cmd == '022': output.append(spointer)
    if cmd == '037' and len(output) > 0: array[apointer] = output.pop(-1)
    if cmd == '038' and len(output) > 0: array[apointer] = output[-1]
    if cmd == '039' and len(output) > 0: array[apointer] = output.pop(0)
    if cmd == '040' and len(output) > 0: array[apointer] = output[0]
    if cmd == '041' and len(output) > 0: output.pop(0)
    if cmd == '042' and len(output) > 0: output.pop(-1)
    if cmd == '172' and apointer != (len(array) - 1):
        output = output + array[apointer+1:]
    if cmd == '173' and apointer != (len(array) - 1):
        data = array[apointer+1:]
        output = output + data
        array = array[0:apointer+1] + [0] * len(data)
    if cmd == '174': output = output + array
    if cmd == '175':
        output = output + array
        array = [0] * len(array)
    return (array, apointer, inputdata, output, source, spointer)
```

```
def logic(array, apointer, inputdata, output, source, spointer):
    '''
    Logical operations

    Instructions handled:
    120: AND operator: Given positive numbers (>0) as True and zero
    or negative numbers (<=0) as False, store Array[current] AND
    Array[current+1] in the current cell (Array[current]) where "0"
    is False and "1" is True.
    121: OR operator: Given positive numbers (>0) as True and zero
    or negative numbers (<=0) as False, store Array[current] OR
    Array[current+1] in the current cell (Array[current]) where "0"
    is False and "1" is True.
    122: NOT operator: Given positive numbers (>0) as True and zero
    or negative numbers (<=0) as False, store NOT Array[current] in
    the current cell (Array[current]) where "0" is False and "1" is True.
    123: LESS-THAN operator: Store Array[current] < Array[current+1]
    in the current cell (Array[current]) where "0" is False and "1" is True.
    124: MORE-THAN operator: Store Array[current] > Array[current+1]
    in the current cell (Array[current]) where "0" is False and "1" is True.
    125: EQUAL operator: Store Array[current] = Array[current+1] in
    the current cell (Array[current]) where "0" is False and "1" is True.
    126: NOT-EQUAL operator: Store Array[current] != Array[current+1]
    in the current cell (Array[current]) where "0" is False and "1" is True.
    127: LESS-THAN-OR-EQUAL operator: Store Array[current]
    <= Array[current+1] in the current cell (Array[current]) where
    "0" is False and "1" is True.
    128: MORE-THAN-OR-EQUAL operator: Store Array[current] =>
    Array[current+1] in the current cell (Array[current]) where "0"
    is False and "1" is True.
    129: NAND operator: Given positive numbers (>0) as True and
    zero or negative numbers (<=0) as False, store Array[current]
    NAND Array[current+1] in the current cell (Array[current])
    where "0" is False and "1" is True. Array[current] NAND
    Array[current+1] is equivalent to NOT (Array[current] AND
    Array[current+1])
    130: NOR operator: Given positive numbers (>0) as True and
    zero or negative numbers (<=0) as False, store Array[current]
    NOR Array[current+1] in the current cell (Array[current]) where
    "0" is False and "1" is True. Array[current] NOR Array[current+1]
    is equivalent to NOT (Array[current] OR Array[current+1])
    '''
    cmd = source[spointer:spointer+3]
    xValue = array[apointer]
    if array[apointer] > 0: x = True
    else: x = False
    if (apointer + 1) < len(array):
        yValue = array[apointer+1]
        if array[apointer+1] > 0: y = True
        else: y = False
    else:
        yValue = array[0]
        if array[0] > 0: y = True
        else: y = False
    if cmd == '120':
        if (x and y) == True: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '121':
        if (x or y) == True: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '122':
        if not x == True: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '123':
        if xValue < yValue: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '124':
        if xValue > yValue: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '125':
        if xValue == yValue: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '126':
        if xValue != yValue: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '127':
        if xValue <= yValue: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '128':
        if xValue >= yValue: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '129':
        if (not (x and y)) == True: array[apointer] = 1
        else: array[apointer] = 0
    if cmd == '130':
        if (not (x or y)) == True: array[apointer] = 1
        else: array[apointer] = 0
    return (array, apointer, inputdata, output, source, spointer)

def flipping(array, apointer, inputdata, output, source, spointer):
    '''
    Flipping of execution elements.

    Instructions handled:
    046: Flip the tape. The original first cell becomes the last
    cell but the tape pointer does not flip in location.
    047: Flip the output list.
    048: Flip the instruction list (source) but the source pointer
    does not flip in location.
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '046': array.reverse()
    if cmd == '047': output.reverse()
    if cmd == '048': source = source[::-1]
    return (array, apointer, inputdata, output, source, spointer)

def input_IO(array, apointer, inputdata, output, source, spointer):
    '''
    Write to and accept values from input list.

    Instructions handled:
    064: Writes the first value of the input list into the current
    cell and without removing the value from the input list. If
    input list is empty, "0" will be written.
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '064':
        if len(inputdata) == 0: array[apointer] = 0
        else: array[apointer] = inputdata[0]
    return (array, apointer, inputdata, output, source, spointer)

def tape_manipulate(array, apointer, inputdata, output, source, spointer):
    '''
    Manipulating the tape.

    Instructions handled:
    081: Swap the value of the current cell (n) and (n+1)th cell.
    133: Flip the tape from the cell after the current cell to the end of the
    tape (temporarily breaking the circularity of the tape).
    161: Cut the tape before the current cell (n) and append it to the end of
    the tape and set tape pointer to 0.
    <---A--->n<---B---> ==> n<---B---><---A--->
    162: Cut the tape after the current cell (n) and append it to the start of
    the tape and set tape pointer to the last cell.
    <---A--->n<---B---> ==> <---B---><---A--->n
    163: Cut out the current cell and append it to the front of the tape and
    set tape pointer to 0. <---A--->n<---B---> ==> n<---A---><---B--->
    164: Cut out the current cell and append it to the end of the tape and
    set tape pointer to the last cell.
    <---A--->n<---B---> ==> <---A---><---B--->n
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '081':
        if (apointer + 1) < len(array):
            temp = array[apointer]
            array[apointer] = array[apointer+1]
            array[apointer+1] = temp
        else:
            temp = array[apointer]
            array[apointer] = array[0]
            array[0] = temp
    if cmd == '133' and (apointer + 1) < len(array):
        temp = array[apointer+1:]
        array = array[0:apointer+1]
        temp.reverse()
        array = array + temp
    if cmd == '161':
        array = array[apointer:] + array[0:apointer]
        apointer = 0
    if cmd == '162':
        array = array[apointer+1:] + array[0:apointer+1]
        apointer = len(array) - 1
    if cmd == '163':
        cell = array.pop(apointer)
        array = [cell] + array
        apointer = 0
    if cmd == '164':
        cell = array.pop(apointer)
        array = array + [cell]
        apointer = len(array) - 1
    return (array, apointer, inputdata, output, source, spointer)

def register_IO(array, apointer, inputdata, output, source, spointer):
    '''
    Read from and write to register from tape cell.

    Instructions handled:
    201: Store value of current tape cell to register #1
    202: Store value of current tape cell to register #2
    203: Store value of current tape cell to register #3
    204: Store value of current tape cell to register #4
    205: Store value of current tape cell to register #5
    206: Store value of current tape cell to register #6
    207: Store value of current tape cell to register #7
    208: Store value of current tape cell to register #8
    209: Store value of current tape cell to register #9
    210: Store value of current tape cell to register #10
    211: Store value of current tape cell to register #11
    212: Store value of current tape cell to register #12
    213: Store value of current tape cell to register #13
    214: Store value of current tape cell to register #14
    215: Store value of current tape cell to register #15
    216: Store value of current tape cell to register #16
    217: Store value of current tape cell to register #17
    218: Store value of current tape cell to register #18
    219: Store value of current tape cell to register #19
    220: Store value of current tape cell to register #20
    221: Store value of current tape cell to register #21
    222: Store value of current tape cell to register #22
    223: Store value of current tape cell to register #23
```

```
224: Store value of current tape cell to register #24
225: Store value of current tape cell to register #25
226: Store value of current tape cell to register #26
227: Store value of current tape cell to register #27
228: Store value of current tape cell to register #28
229: Store value of current tape cell to register #29
230: Store value of current tape cell to register #30
231: Store value of current tape cell to register #31
232: Store value of current tape cell to register #32
233: Store value of current tape cell to register #33
234: Store value of current tape cell to register #34
235: Store value of current tape cell to register #35
236: Store value of current tape cell to register #36
237: Store value of current tape cell to register #37
238: Store value of current tape cell to register #38
239: Store value of current tape cell to register #39
240: Store value of current tape cell to register #40
241: Store value of current tape cell to register #41
242: Store value of current tape cell to register #42
243: Store value of current tape cell to register #43
244: Store value of current tape cell to register #44
245: Store value of current tape cell to register #45
246: Store value of current tape cell to register #46
247: Store value of current tape cell to register #47
248: Store value of current tape cell to register #48
249: Store value of current tape cell to register #49
250: Store value of current tape cell to register #50
251: Store value of current tape cell to register #51
252: Store value of current tape cell to register #52
253: Store value of current tape cell to register #53
254: Store value of current tape cell to register #54
255: Store value of current tape cell to register #55
256: Store value of current tape cell to register #56
257: Store value of current tape cell to register #57
258: Store value of current tape cell to register #58
259: Store value of current tape cell to register #59
260: Store value of current tape cell to register #60
261: Store value of current tape cell to register #61
262: Store value of current tape cell to register #62
263: Store value of current tape cell to register #63
264: Store value of current tape cell to register #64
265: Store value of current tape cell to register #65
266: Store value of current tape cell to register #66
267: Store value of current tape cell to register #67
268: Store value of current tape cell to register #68
269: Store value of current tape cell to register #69
270: Store value of current tape cell to register #70
271: Store value of current tape cell to register #71
272: Store value of current tape cell to register #72
273: Store value of current tape cell to register #73
274: Store value of current tape cell to register #74
275: Store value of current tape cell to register #75
276: Store value of current tape cell to register #76
277: Store value of current tape cell to register #77
278: Store value of current tape cell to register #78
279: Store value of current tape cell to register #79
280: Store value of current tape cell to register #80
281: Store value of current tape cell to register #81
282: Store value of current tape cell to register #82
283: Store value of current tape cell to register #83
284: Store value of current tape cell to register #84
285: Store value of current tape cell to register #85
286: Store value of current tape cell to register #86
287: Store value of current tape cell to register #87
288: Store value of current tape cell to register #88
289: Store value of current tape cell to register #89
290: Store value of current tape cell to register #90
```

```
291: Store value of current tape cell to register #91
292: Store value of current tape cell to register #92
293: Store value of current tape cell to register #93
294: Store value of current tape cell to register #94
295: Store value of current tape cell to register #95
296: Store value of current tape cell to register #96
297: Store value of current tape cell to register #97
298: Store value of current tape cell to register #98
299: Store value of current tape cell to register #99
301: Put value from register #1 to current tape cell
302: Put value from register #2 to current tape cell
303: Put value from register #3 to current tape cell
304: Put value from register #4 to current tape cell
305: Put value from register #5 to current tape cell
306: Put value from register #6 to current tape cell
307: Put value from register #7 to current tape cell
308: Put value from register #8 to current tape cell
309: Put value from register #9 to current tape cell
310: Put value from register #10 to current tape cell
311: Put value from register #11 to current tape cell
312: Put value from register #12 to current tape cell
313: Put value from register #13 to current tape cell
314: Put value from register #14 to current tape cell
315: Put value from register #15 to current tape cell
316: Put value from register #16 to current tape cell
317: Put value from register #17 to current tape cell
318: Put value from register #18 to current tape cell
319: Put value from register #19 to current tape cell
320: Put value from register #20 to current tape cell
321: Put value from register #21 to current tape cell
322: Put value from register #22 to current tape cell
323: Put value from register #23 to current tape cell
324: Put value from register #24 to current tape cell
325: Put value from register #25 to current tape cell
326: Put value from register #26 to current tape cell
327: Put value from register #27 to current tape cell
328: Put value from register #28 to current tape cell
329: Put value from register #29 to current tape cell
330: Put value from register #30 to current tape cell
331: Put value from register #31 to current tape cell
332: Put value from register #32 to current tape cell
333: Put value from register #33 to current tape cell
334: Put value from register #34 to current tape cell
335: Put value from register #35 to current tape cell
336: Put value from register #36 to current tape cell
337: Put value from register #37 to current tape cell
338: Put value from register #38 to current tape cell
339: Put value from register #39 to current tape cell
340: Put value from register #40 to current tape cell
341: Put value from register #41 to current tape cell
342: Put value from register #42 to current tape cell
343: Put value from register #43 to current tape cell
344: Put value from register #44 to current tape cell
345: Put value from register #45 to current tape cell
346: Put value from register #46 to current tape cell
347: Put value from register #47 to current tape cell
348: Put value from register #48 to current tape cell
349: Put value from register #49 to current tape cell
350: Put value from register #50 to current tape cell
351: Put value from register #51 to current tape cell
352: Put value from register #52 to current tape cell
353: Put value from register #53 to current tape cell
354: Put value from register #54 to current tape cell
355: Put value from register #55 to current tape cell
356: Put value from register #56 to current tape cell
357: Put value from register #57 to current tape cell
358: Put value from register #58 to current tape cell
```

```
359: Put value from register #59 to current tape cell
360: Put value from register #60 to current tape cell
361: Put value from register #61 to current tape cell
362: Put value from register #62 to current tape cell
363: Put value from register #63 to current tape cell
364: Put value from register #64 to current tape cell
365: Put value from register #65 to current tape cell
366: Put value from register #66 to current tape cell
367: Put value from register #67 to current tape cell
368: Put value from register #68 to current tape cell
369: Put value from register #69 to current tape cell
370: Put value from register #70 to current tape cell
371: Put value from register #71 to current tape cell
372: Put value from register #72 to current tape cell
373: Put value from register #73 to current tape cell
374: Put value from register #74 to current tape cell
375: Put value from register #75 to current tape cell
376: Put value from register #76 to current tape cell
377: Put value from register #77 to current tape cell
378: Put value from register #78 to current tape cell
379: Put value from register #79 to current tape cell
380: Put value from register #80 to current tape cell
381: Put value from register #81 to current tape cell
382: Put value from register #82 to current tape cell
383: Put value from register #83 to current tape cell
384: Put value from register #84 to current tape cell
385: Put value from register #85 to current tape cell
386: Put value from register #86 to current tape cell
387: Put value from register #87 to current tape cell
388: Put value from register #88 to current tape cell
389: Put value from register #89 to current tape cell
390: Put value from register #90 to current tape cell
391: Put value from register #91 to current tape cell
392: Put value from register #92 to current tape cell
393: Put value from register #93 to current tape cell
394: Put value from register #94 to current tape cell
395: Put value from register #95 to current tape cell
396: Put value from register #96 to current tape cell
397: Put value from register #97 to current tape cell
398: Put value from register #98 to current tape cell
399: Put value from register #99 to current tape cell
'''
cmd = source[spointer:spointer+3]
if cmd == '201': register[0] = array[apointer]
if cmd == '202': register[1] = array[apointer]
if cmd == '203': register[2] = array[apointer]
if cmd == '204': register[3] = array[apointer]
if cmd == '205': register[4] = array[apointer]
if cmd == '206': register[5] = array[apointer]
if cmd == '207': register[6] = array[apointer]
if cmd == '208': register[7] = array[apointer]
if cmd == '209': register[8] = array[apointer]
if cmd == '210': register[9] = array[apointer]
if cmd == '211': register[10] = array[apointer]
if cmd == '212': register[11] = array[apointer]
if cmd == '213': register[12] = array[apointer]
if cmd == '214': register[13] = array[apointer]
if cmd == '215': register[14] = array[apointer]
if cmd == '216': register[15] = array[apointer]
if cmd == '217': register[16] = array[apointer]
if cmd == '218': register[17] = array[apointer]
if cmd == '219': register[18] = array[apointer]
if cmd == '220': register[19] = array[apointer]
if cmd == '221': register[20] = array[apointer]
if cmd == '222': register[21] = array[apointer]
if cmd == '223': register[22] = array[apointer]
if cmd == '224': register[23] = array[apointer]
if cmd == '225': register[24] = array[apointer]
if cmd == '226': register[25] = array[apointer]
if cmd == '227': register[26] = array[apointer]
if cmd == '228': register[27] = array[apointer]
if cmd == '229': register[28] = array[apointer]
if cmd == '230': register[29] = array[apointer]
if cmd == '231': register[30] = array[apointer]
if cmd == '232': register[31] = array[apointer]
if cmd == '233': register[32] = array[apointer]
if cmd == '234': register[33] = array[apointer]
if cmd == '235': register[34] = array[apointer]
if cmd == '236': register[35] = array[apointer]
if cmd == '237': register[36] = array[apointer]
if cmd == '238': register[37] = array[apointer]
if cmd == '239': register[38] = array[apointer]
if cmd == '240': register[39] = array[apointer]
if cmd == '241': register[40] = array[apointer]
if cmd == '242': register[41] = array[apointer]
if cmd == '243': register[42] = array[apointer]
if cmd == '244': register[43] = array[apointer]
if cmd == '245': register[44] = array[apointer]
if cmd == '246': register[45] = array[apointer]
if cmd == '247': register[46] = array[apointer]
if cmd == '248': register[47] = array[apointer]
if cmd == '249': register[48] = array[apointer]
if cmd == '250': register[49] = array[apointer]
if cmd == '251': register[50] = array[apointer]
if cmd == '252': register[51] = array[apointer]
if cmd == '253': register[52] = array[apointer]
if cmd == '254': register[53] = array[apointer]
if cmd == '255': register[54] = array[apointer]
if cmd == '256': register[55] = array[apointer]
if cmd == '257': register[56] = array[apointer]
if cmd == '258': register[57] = array[apointer]
if cmd == '259': register[58] = array[apointer]
if cmd == '260': register[59] = array[apointer]
if cmd == '261': register[60] = array[apointer]
if cmd == '262': register[61] = array[apointer]
if cmd == '263': register[62] = array[apointer]
if cmd == '264': register[63] = array[apointer]
if cmd == '265': register[64] = array[apointer]
if cmd == '266': register[65] = array[apointer]
if cmd == '267': register[66] = array[apointer]
if cmd == '268': register[67] = array[apointer]
if cmd == '269': register[68] = array[apointer]
if cmd == '270': register[69] = array[apointer]
if cmd == '271': register[70] = array[apointer]
if cmd == '272': register[71] = array[apointer]
if cmd == '273': register[72] = array[apointer]
if cmd == '274': register[73] = array[apointer]
if cmd == '275': register[74] = array[apointer]
if cmd == '276': register[75] = array[apointer]
if cmd == '277': register[76] = array[apointer]
if cmd == '278': register[77] = array[apointer]
if cmd == '279': register[78] = array[apointer]
if cmd == '280': register[79] = array[apointer]
if cmd == '281': register[80] = array[apointer]
if cmd == '282': register[81] = array[apointer]
if cmd == '283': register[82] = array[apointer]
if cmd == '284': register[83] = array[apointer]
if cmd == '285': register[84] = array[apointer]
if cmd == '286': register[85] = array[apointer]
if cmd == '287': register[86] = array[apointer]
if cmd == '288': register[87] = array[apointer]
if cmd == '289': register[88] = array[apointer]
if cmd == '290': register[89] = array[apointer]
if cmd == '291': register[90] = array[apointer]
if cmd == '292': register[91] = array[apointer]
if cmd == '293': register[92] = array[apointer]
if cmd == '294': register[93] = array[apointer]
if cmd == '295': register[94] = array[apointer]
if cmd == '296': register[95] = array[apointer]
if cmd == '297': register[96] = array[apointer]
if cmd == '298': register[97] = array[apointer]
if cmd == '299': register[98] = array[apointer]
if cmd == '301': array[apointer] = register[0]
if cmd == '302': array[apointer] = register[1]
if cmd == '303': array[apointer] = register[2]
if cmd == '304': array[apointer] = register[3]
if cmd == '305': array[apointer] = register[4]
if cmd == '306': array[apointer] = register[5]
if cmd == '307': array[apointer] = register[6]
if cmd == '308': array[apointer] = register[7]
if cmd == '309': array[apointer] = register[8]
if cmd == '310': array[apointer] = register[9]
if cmd == '311': array[apointer] = register[10]
if cmd == '312': array[apointer] = register[11]
if cmd == '313': array[apointer] = register[12]
if cmd == '314': array[apointer] = register[13]
if cmd == '315': array[apointer] = register[14]
if cmd == '316': array[apointer] = register[15]
if cmd == '317': array[apointer] = register[16]
if cmd == '318': array[apointer] = register[17]
if cmd == '319': array[apointer] = register[18]
if cmd == '320': array[apointer] = register[19]
if cmd == '321': array[apointer] = register[20]
if cmd == '322': array[apointer] = register[21]
if cmd == '323': array[apointer] = register[22]
if cmd == '324': array[apointer] = register[23]
if cmd == '325': array[apointer] = register[24]
if cmd == '326': array[apointer] = register[25]
if cmd == '327': array[apointer] = register[26]
if cmd == '328': array[apointer] = register[27]
if cmd == '329': array[apointer] = register[28]
if cmd == '330': array[apointer] = register[29]
if cmd == '331': array[apointer] = register[30]
if cmd == '332': array[apointer] = register[31]
if cmd == '333': array[apointer] = register[32]
if cmd == '334': array[apointer] = register[33]
if cmd == '335': array[apointer] = register[34]
if cmd == '336': array[apointer] = register[35]
if cmd == '337': array[apointer] = register[36]
if cmd == '338': array[apointer] = register[37]
if cmd == '339': array[apointer] = register[38]
if cmd == '340': array[apointer] = register[39]
if cmd == '341': array[apointer] = register[40]
if cmd == '342': array[apointer] = register[41]
if cmd == '343': array[apointer] = register[42]
if cmd == '344': array[apointer] = register[43]
if cmd == '345': array[apointer] = register[44]
if cmd == '346': array[apointer] = register[45]
if cmd == '347': array[apointer] = register[46]
if cmd == '348': array[apointer] = register[47]
if cmd == '349': array[apointer] = register[48]
if cmd == '350': array[apointer] = register[49]
if cmd == '351': array[apointer] = register[50]
if cmd == '352': array[apointer] = register[51]
if cmd == '353': array[apointer] = register[52]
if cmd == '354': array[apointer] = register[53]
if cmd == '355': array[apointer] = register[54]
if cmd == '356': array[apointer] = register[55]
if cmd == '357': array[apointer] = register[56]
if cmd == '358': array[apointer] = register[57]
if cmd == '359': array[apointer] = register[58]
```

```
    if cmd == '360': array[apointer] = register[59]
    if cmd == '361': array[apointer] = register[60]
    if cmd == '362': array[apointer] = register[61]
    if cmd == '363': array[apointer] = register[62]
    if cmd == '364': array[apointer] = register[63]
    if cmd == '365': array[apointer] = register[64]
    if cmd == '366': array[apointer] = register[65]
    if cmd == '367': array[apointer] = register[66]
    if cmd == '368': array[apointer] = register[67]
    if cmd == '369': array[apointer] = register[68]
    if cmd == '370': array[apointer] = register[69]
    if cmd == '371': array[apointer] = register[70]
    if cmd == '372': array[apointer] = register[71]
    if cmd == '373': array[apointer] = register[72]
    if cmd == '374': array[apointer] = register[73]
    if cmd == '375': array[apointer] = register[74]
    if cmd == '376': array[apointer] = register[75]
    if cmd == '377': array[apointer] = register[76]
    if cmd == '378': array[apointer] = register[77]
    if cmd == '379': array[apointer] = register[78]
    if cmd == '380': array[apointer] = register[79]
    if cmd == '381': array[apointer] = register[80]
    if cmd == '382': array[apointer] = register[81]
    if cmd == '383': array[apointer] = register[82]
    if cmd == '384': array[apointer] = register[83]
    if cmd == '385': array[apointer] = register[84]
    if cmd == '386': array[apointer] = register[85]
    if cmd == '387': array[apointer] = register[86]
    if cmd == '388': array[apointer] = register[87]
    if cmd == '389': array[apointer] = register[88]
    if cmd == '390': array[apointer] = register[89]
    if cmd == '391': array[apointer] = register[90]
    if cmd == '392': array[apointer] = register[91]
    if cmd == '393': array[apointer] = register[92]
    if cmd == '394': array[apointer] = register[93]
    if cmd == '395': array[apointer] = register[94]
    if cmd == '396': array[apointer] = register[95]
    if cmd == '397': array[apointer] = register[96]
    if cmd == '398': array[apointer] = register[97]
    if cmd == '399': array[apointer] = register[98]
    return (array, apointer, inputdata, output, source, spointer)

def jump_identifier(array, apointer, inputdata, output, source, spointer):
    '''
    Defines jump location within the source tape. These instructions acts as
    Pure identifiers and do not perform any operations.

    Instructions handled:
    200, 300, 400, 500, 600, 700, 800, 900
    '''
    cmd = source[spointer:spointer+3]
    if cmd == '200': pass
    if cmd == '300': pass
    if cmd == '400': pass
    if cmd == '500': pass
    if cmd == '600': pass
    if cmd == '700': pass
    if cmd == '800': pass
    if cmd == '900': pass
    return (array, apointer, inputdata, output, source, spointer)

def not_used(array, apointer, inputdata, output, source, spointer):
    '''
    Default do-nothing handler for not implemented instructions.
    '''
    return (array, apointer, inputdata, output, source, spointer)
```

```
ragaraja = {'000': forward, '001': tape_move, '002': tape_move, '003': tape_move,
            '004': backward, '005': tape_move, '006': tape_move, '007': tape_move,
            '008': increment, '009': accumulations,
            '010': accumulations, '011': decrement,
            '012': accumulations, '013': accumulations,
            '014': loop_start, '015': loop_end,
            '016': tape_size, '017': tape_size, '018': tape_size, '019': tape_size,
            '020': call_out, '021': output_IO, '022': output_IO,
            '023': source_move, '024': source_move, '025': source_move,
            '026': source_move, '027': source_move, '028': source_move,
            '029': not_used, '030': not_used, '031': not_used,
            '032': accumulations, '033': accumulations,
            '034': tape_size, '035': tape_size, '036': tape_size,
            '037': output_IO, '038': output_IO, '039': output_IO,
            '040': output_IO, '041': output_IO, '042': output_IO,
            '043': tape_move,'044': tape_move, '045': tape_move,
            '046': flipping, '047': flipping, '048': flipping, '049': not_used,
            '050': nBF_random_op, '051': nBF_random_op, '052': nBF_random_op,
            '053': nBF_random_op, '054': nBF_random_op, '055': nBF_random_op,
            '056': nBF_random_op, '057': nBF_random_op, '058': nBF_random_op,
            '059': nBF_random_op, '060': nBF_random_op, '061': tape_move,
            '062': tape_move, '063': accept_predefined,
            '064': input_IO, '065': mathematics,
            '066': mathematics, '067': mathematics, '068': mathematics,
            '069': mathematics, '070': mathematics, '071': mathematics,
            '072': mathematics, '073': mathematics, '074': mathematics,
            '075': mathematics, '076': mathematics, '077': mathematics,
            '078': mathematics, '079': mathematics,
            '080': mathematics, '081': tape_manipulate,
            '082': source_move, '083': source_move,
            '084': set_tape_value, '085': set_tape_value,
            '086': set_tape_value, '087': mathematics,
            '088': mathematics, '089': mathematics, '090': mathematics,
            '091': mathematics, '092': mathematics, '093': mathematics,
            '094': mathematics, '095': mathematics,
            '096': mathematics, '097': set_tape_value,
            '098': set_tape_value, '099': mathematics,
            '100': mathematics, '101': mathematics, '102': mathematics,
            '103': mathematics, '104': mathematics, '105': mathematics,
            '106': mathematics, '107': mathematics, '108': mathematics,
            '109': mathematics, '110': mathematics, '111': mathematics,
            '112': mathematics, '113': mathematics, '114': mathematics,
            '115': mathematics, '116': mathematics, '117': mathematics,
            '118': not_used, '119': not_used,
            '120': logic, '121': logic, '122': logic, '123': logic,
            '124': logic, '125': logic, '126': logic, '127': logic,
            '128': logic, '129': logic, '130': logic, '131': not_used,
            '132': not_used, '133': tape_manipulate,
            '134': not_used, '135': not_used, '136': not_used, '137': not_used,
            '138': not_used, '139': not_used,
            '140': tape_move, '141': tape_move, '142': tape_move, '143': tape_move,
            '144': mathematics, '145': mathematics, '146': mathematics,
            '147': mathematics, '148': mathematics, '149': mathematics,
            '150': mathematics, '151': mathematics, '152': mathematics,
            '153': mathematics, '154': mathematics, '155': mathematics,
            '156': mathematics, '157': mathematics, '158': mathematics,
            '159': mathematics, '160': mathematics, '161': tape_manipulate,
            '162': tape_manipulate, '163': tape_manipulate, '164': tape_manipulate,
            '165': mathematics, '166': mathematics, '167': mathematics,
            '168': mathematics, '169': mathematics, '170': mathematics,
            '171': mathematics, '172': output_IO, '173': output_IO,
            '174': output_IO, '175': output_IO,
            '176': not_used, '177': not_used, '178': not_used, '179': not_used,
            '180': not_used, '181': not_used, '182': not_used, '183': not_used,
            '184': not_used, '185': not_used, '186': not_used,
            '187': set_tape_value, '188': set_tape_value, '189': set_tape_value,
            '190': set_tape_value, '191': set_tape_value, '192': set_tape_value,
            '193': set_tape_value, '194': set_tape_value, '195': not_used,
            '196': mathematics, '197': mathematics, '198': mathematics,
            '199': not_used, '200': jump_identifier, '201': register_IO,
            '202': register_IO, '203': register_IO, '204': register_IO,
            '205': register_IO, '206': register_IO, '207': register_IO,
            '208': register_IO, '209': register_IO, '210': register_IO,
            '211': register_IO, '212': register_IO, '213': register_IO,
            '214': register_IO, '215': register_IO, '216': register_IO,
            '217': register_IO, '218': register_IO, '219': register_IO,
            '220': register_IO, '221': register_IO, '222': register_IO,
            '223': register_IO, '224': register_IO, '225': register_IO,
            '226': register_IO, '227': register_IO, '228': register_IO,
            '229': register_IO, '230': register_IO, '231': register_IO,
            '232': register_IO, '233': register_IO, '234': register_IO,
            '235': register_IO, '236': register_IO, '237': register_IO,
            '238': register_IO, '239': register_IO, '240': register_IO,
            '241': register_IO, '242': register_IO, '243': register_IO,
            '244': register_IO, '245': register_IO, '246': register_IO,
            '247': register_IO, '248': register_IO, '249': register_IO,
            '250': register_IO, '251': register_IO, '252': register_IO,
            '253': register_IO, '254': register_IO, '255': register_IO,
            '256': register_IO, '257': register_IO, '258': register_IO,
            '259': register_IO, '260': register_IO, '261': register_IO,
            '262': register_IO, '263': register_IO, '264': register_IO,
            '265': register_IO, '266': register_IO, '267': register_IO,
            '268': register_IO, '269': register_IO, '270': register_IO,
            '271': register_IO, '272': register_IO, '273': register_IO,
            '274': register_IO, '275': register_IO, '276': register_IO,
            '277': register_IO, '278': register_IO, '279': register_IO,
            '280': register_IO, '281': register_IO, '282': register_IO,
            '283': register_IO, '284': register_IO, '285': register_IO,
            '286': register_IO, '287': register_IO, '288': register_IO,
            '289': register_IO, '290': register_IO, '291': register_IO,
            '292': register_IO, '293': register_IO, '294': register_IO,
            '295': register_IO, '296': register_IO, '297': register_IO,
            '298': register_IO, '299': register_IO, '300': jump_identifier,
            '301': register_IO, '302': register_IO, '303': register_IO,
            '304': register_IO, '305': register_IO, '306': register_IO,
            '307': register_IO, '308': register_IO, '309': register_IO,
            '310': register_IO, '311': register_IO, '312': register_IO,
            '313': register_IO, '314': register_IO, '315': register_IO,
            '316': register_IO, '317': register_IO, '318': register_IO,
            '319': register_IO, '320': register_IO, '321': register_IO,
            '322': register_IO, '323': register_IO, '324': register_IO,
            '325': register_IO, '326': register_IO, '327': register_IO,
            '328': register_IO, '329': register_IO, '330': register_IO,
            '331': register_IO, '332': register_IO, '333': register_IO,
            '334': register_IO, '335': register_IO, '336': register_IO,
            '337': register_IO, '338': register_IO, '339': register_IO,
            '340': register_IO, '341': register_IO, '342': register_IO,
            '343': register_IO, '344': register_IO, '345': register_IO,
            '346': register_IO, '347': register_IO, '348': register_IO,
            '349': register_IO, '350': register_IO, '351': register_IO,
            '352': register_IO, '353': register_IO, '354': register_IO,
            '355': register_IO, '356': register_IO, '357': register_IO,
            '358': register_IO, '359': register_IO, '360': register_IO,
            '361': register_IO, '362': register_IO, '363': register_IO,
            '364': register_IO, '365': register_IO, '366': register_IO,
            '367': register_IO, '368': register_IO, '369': register_IO,
            '370': register_IO, '371': register_IO, '372': register_IO,
            '373': register_IO, '374': register_IO, '375': register_IO,
            '376': register_IO, '377': register_IO, '378': register_IO,
            '379': register_IO, '380': register_IO, '381': register_IO,
            '382': register_IO, '383': register_IO, '384': register_IO,
            '385': register_IO, '386': register_IO, '387': register_IO,
            '388': register_IO, '389': register_IO, '390': register_IO,
            '391': register_IO, '392': register_IO, '393': register_IO,
            '394': register_IO, '395': register_IO, '396': register_IO,
            '397': register_IO, '398': register_IO, '399': register_IO,
            '400': jump_identifier, '401': not_used,
            '402': not_used, '403': not_used, '404': not_used, '405': not_used,
            '406': not_used, '407': not_used, '408': not_used, '409': not_used,
            '410': not_used, '411': not_used, '412': not_used, '413': not_used,
            '414': not_used, '415': not_used, '416': not_used, '417': not_used,
            '418': not_used, '419': not_used, '420': not_used, '421': not_used,
            '422': not_used, '423': not_used, '424': not_used, '425': not_used,
            '426': not_used, '427': not_used, '428': not_used, '429': not_used,
            '430': not_used, '431': not_used, '432': not_used, '433': not_used,
            '434': not_used, '435': not_used, '436': not_used, '437': not_used,
            '438': not_used, '439': not_used, '440': not_used, '441': not_used,
            '442': not_used, '443': not_used, '444': not_used, '445': not_used,
            '446': not_used, '447': not_used, '448': not_used, '449': not_used,
            '450': not_used, '451': not_used, '452': not_used, '453': not_used,
            '454': not_used, '455': not_used, '456': not_used, '457': not_used,
            '458': not_used, '459': not_used, '460': not_used, '461': not_used,
            '462': not_used, '463': not_used, '464': not_used, '465': not_used,
            '466': not_used, '467': not_used, '468': not_used, '469': not_used,
            '470': not_used, '471': not_used, '472': not_used, '473': not_used,
            '474': not_used, '475': not_used, '476': not_used, '477': not_used,
            '478': not_used, '479': not_used, '480': not_used, '481': not_used,
            '482': not_used, '483': not_used, '484': not_used, '485': not_used,
            '486': not_used, '487': not_used, '488': not_used, '489': not_used,
            '490': not_used, '491': not_used, '492': not_used, '493': not_used,
            '494': not_used, '495': not_used, '496': not_used, '497': not_used,
            '498': not_used, '499': not_used,
            '500': jump_identifier, '501': not_used,
            '502': not_used, '503': not_used, '504': not_used, '505': not_used,
            '506': not_used, '507': not_used, '508': not_used, '509': not_used,
            '510': not_used, '511': not_used, '512': not_used, '513': not_used,
            '514': not_used, '515': not_used, '516': not_used, '517': not_used,
            '518': not_used, '519': not_used, '520': not_used, '521': not_used,
            '522': not_used, '523': not_used, '524': not_used, '525': not_used,
            '526': not_used, '527': not_used, '528': not_used, '529': not_used,
            '530': not_used, '531': not_used, '532': not_used, '533': not_used,
            '534': not_used, '535': not_used, '536': not_used, '537': not_used,
            '538': not_used, '539': not_used, '540': not_used, '541': not_used,
            '542': not_used, '543': not_used, '544': not_used, '545': not_used,
            '546': not_used, '547': not_used, '548': not_used, '549': not_used,
            '550': not_used, '551': not_used, '552': not_used, '553': not_used,
            '554': not_used, '555': not_used, '556': not_used, '557': not_used,
            '558': not_used, '559': not_used, '560': not_used, '561': not_used,
            '562': not_used, '563': not_used, '564': not_used, '565': not_used,
            '566': not_used, '567': not_used, '568': not_used, '569': not_used,
            '570': not_used, '571': not_used, '572': not_used, '573': not_used,
            '574': not_used, '575': not_used, '576': not_used, '577': not_used,
            '578': not_used, '579': not_used, '580': not_used, '581': not_used,
            '582': not_used, '583': not_used, '584': not_used, '585': not_used,
            '586': not_used, '587': not_used, '588': not_used, '589': not_used,
            '590': not_used, '591': not_used, '592': not_used, '593': not_used,
            '594': not_used, '595': not_used, '596': not_used, '597': not_used,
            '598': not_used, '599': not_used,
            '600': jump_identifier, '601': not_used,
            '602': not_used, '603': not_used, '604': not_used, '605': not_used,
            '606': not_used, '607': not_used, '608': not_used, '609': not_used,
            '610': not_used, '611': not_used, '612': not_used, '613': not_used,
            '614': not_used, '615': not_used, '616': not_used, '617': not_used,
            '618': not_used, '619': not_used, '620': not_used, '621': not_used,
            '622': not_used, '623': not_used, '624': not_used, '625': not_used,
            '626': not_used, '627': not_used, '628': not_used, '629': not_used,
            '630': not_used, '631': not_used, '632': not_used, '633': not_used,
            '634': not_used, '635': not_used, '636': not_used, '637': not_used,
            '638': not_used, '639': not_used, '640': not_used, '641': not_used,
            '642': not_used, '643': not_used, '644': not_used, '645': not_used,
            '646': not_used, '647': not_used, '648': not_used, '649': not_used,
            '650': not_used, '651': not_used, '652': not_used, '653': not_used,
            '654': not_used, '655': not_used, '656': not_used, '657': not_used,
            '658': not_used, '659': not_used, '660': not_used, '661': not_used,
            '662': not_used, '663': not_used, '664': not_used, '665': not_used,
            '666': not_used, '667': not_used, '668': not_used, '669': not_used,
            '670': not_used, '671': not_used, '672': not_used, '673': not_used,
            '674': not_used, '675': not_used, '676': not_used, '677': not_used,
            '678': not_used, '679': not_used, '680': not_used, '681': not_used,
            '682': not_used, '683': not_used, '684': not_used, '685': not_used,
            '686': not_used, '687': not_used, '688': not_used, '689': not_used,
            '690': not_used, '691': not_used, '692': not_used, '693': not_used,
            '694': not_used, '695': not_used, '696': not_used, '697': not_used,
            '698': not_used, '699': not_used,
            '700': jump_identifier, '701': not_used,
            '702': not_used, '703': not_used, '704': not_used, '705': not_used,
            '706': not_used, '707': not_used, '708': not_used, '709': not_used,
            '710': not_used, '711': not_used, '712': not_used, '713': not_used,
            '714': not_used, '715': not_used, '716': not_used, '717': not_used,
            '718': not_used, '719': not_used, '720': not_used, '721': not_used,
            '722': not_used, '723': not_used, '724': not_used, '725': not_used,
            '726': not_used, '727': not_used, '728': not_used, '729': not_used,
            '730': not_used, '731': not_used, '732': not_used, '733': not_used,
            '734': not_used, '735': not_used, '736': not_used, '737': not_used,
            '738': not_used, '739': not_used, '740': not_used, '741': not_used,
            '742': not_used, '743': not_used, '744': not_used, '745': not_used,
            '746': not_used, '747': not_used, '748': not_used, '749': not_used,
            '750': not_used, '751': not_used, '752': not_used, '753': not_used,
            '754': not_used, '755': not_used, '756': not_used, '757': not_used,
            '758': not_used, '759': not_used, '760': not_used, '761': not_used,
            '762': not_used, '763': not_used, '764': not_used, '765': not_used,
            '766': not_used, '767': not_used, '768': not_used, '769': not_used,
            '770': not_used, '771': not_used, '772': not_used, '773': not_used,
            '774': not_used, '775': not_used, '776': not_used, '777': not_used,
            '778': not_used, '779': not_used, '780': not_used, '781': not_used,
            '782': not_used, '783': not_used, '784': not_used, '785': not_used,
            '786': not_used, '787': not_used, '788': not_used, '789': not_used,
            '790': not_used, '791': not_used, '792': not_used, '793': not_used,
            '794': not_used, '795': not_used, '796': not_used, '797': not_used,
            '798': not_used, '799': not_used,
            '800': jump_identifier, '801': not_used,
            '802': not_used, '803': not_used, '804': not_used, '805': not_used,
            '806': not_used, '807': not_used, '808': not_used, '809': not_used,
            '810': not_used, '811': not_used, '812': not_used, '813': not_used,
            '814': not_used, '815': not_used, '816': not_used, '817': not_used,
            '818': not_used, '819': not_used, '820': not_used, '821': not_used,
            '822': not_used, '823': not_used, '824': not_used, '825': not_used,
            '826': not_used, '827': not_used, '828': not_used, '829': not_used,
            '830': not_used, '831': not_used, '832': not_used, '833': not_used,
            '834': not_used, '835': not_used, '836': not_used, '837': not_used,
            '838': not_used, '839': not_used, '840': not_used, '841': not_used,
            '842': not_used, '843': not_used, '844': not_used, '845': not_used,
            '846': not_used, '847': not_used, '848': not_used, '849': not_used,
            '850': not_used, '851': not_used, '852': not_used, '853': not_used,
            '854': not_used, '855': not_used, '856': not_used, '857': not_used,
            '858': not_used, '859': not_used, '860': not_used, '861': not_used,
            '862': not_used, '863': not_used, '864': not_used, '865': not_used,
            '866': not_used, '867': not_used, '868': not_used, '869': not_used,
            '870': not_used, '871': not_used, '872': not_used, '873': not_used,
            '874': not_used, '875': not_used, '876': not_used, '877': not_used,
            '878': not_used, '879': not_used, '880': not_used, '881': not_used,
            '882': not_used, '883': not_used, '884': not_used, '885': not_used,
            '886': not_used, '887': not_used, '888': not_used, '889': not_used,
            '890': not_used, '891': not_used, '892': not_used, '893': not_used,
            '894': not_used, '895': not_used, '896': not_used, '897': not_used,
            '898': not_used, '899': not_used,
            '900': jump_identifier, '901': not_used,
            '902': not_used, '903': not_used, '904': not_used, '905': not_used,
            '906': not_used, '907': not_used, '908': not_used, '909': not_used,
            '910': not_used, '911': not_used, '912': not_used, '913': not_used,
            '914': not_used, '915': not_used, '916': not_used, '917': not_used,
            '918': not_used, '919': not_used, '920': not_used, '921': not_used,
            '922': not_used, '923': not_used, '924': not_used, '925': not_used,
            '926': not_used, '927': not_used, '928': not_used, '929': not_used,
            '930': not_used, '931': not_used, '932': not_used, '933': not_used,
            '934': not_used, '935': not_used, '936': not_used, '937': not_used,
            '938': not_used, '939': not_used, '940': not_used, '941': not_used,
            '942': not_used, '943': not_used, '944': not_used, '945': not_used,
            '946': not_used, '947': not_used, '948': not_used, '949': not_used,
            '950': not_used, '951': not_used, '952': not_used, '953': not_used,
            '954': not_used, '955': not_used, '956': not_used, '957': not_used,
            '958': not_used, '959': not_used, '960': not_used, '961': not_used,
            '962': not_used, '963': not_used, '964': not_used, '965': not_used,
            '966': not_used, '967': not_used, '968': not_used, '969': not_used,
            '970': not_used, '971': not_used, '972': not_used, '973': not_used,
            '974': not_used, '975': not_used, '976': not_used, '977': not_used,
            '978': not_used, '979': not_used, '980': not_used, '981': not_used,
            '982': not_used, '983': not_used, '984': not_used, '985': not_used,
            '986': not_used, '987': not_used, '988': not_used, '989': not_used,
            '990': not_used, '991': not_used, '992': not_used, '993': not_used,
            '994': not_used, '995': not_used, '996': not_used, '997': not_used,
            '998': not_used, '999': not_used}

ragaraja_v1 = [
    '000', '001', '002', '003', '004', '005', '006', '007', '008', '009',
    '010', '011', '012', '013', '016', '017', '018', '019',
    '020', '021', '022', '023', '024', '025',
    '032', '033', '034', '035', '036', '037', '038', '039',
    '040', '041', '042', '043', '044', '045', '046', '047',
    '050', '051', '052', '053', '054', '055', '056', '057', '058', '059',
    '060', '061', '062', '063', '064', '065', '066', '067', '068', '069',
    '070', '071', '072', '073', '074', '075', '076', '077', '078', '079',
    '080', '081', '084', '085', '086', '087', '088', '089',
    '090', '091', '092', '093', '094', '095', '096', '097', '098', '099',
    '100', '101', '102', '103', '104', '105', '106', '107', '108', '109',
    '110', '111', '112', '113', '114', '115', '116', '117',
    '120', '121', '122', '123', '124', '125', '126', '127', '128',
    '140', '141', '142', '143', '144', '145', '146', '147',
    '150', '151', '152', '153', '154', '155', '156', '157', '158', '159',
    '160', '161', '162', '163', '164', '165', '166', '167', '168', '169',
    '170', '171',
    '189',
    '196', '197', '198',
    '201', '202', '203', '204', '205', '206', '207', '208', '209',
    '210', '211', '212', '213', '214', '215', '216', '217', '218', '219',
    '220', '221', '222', '223', '224', '225', '226', '227', '228', '229',
    '230', '231', '232', '233', '234', '235', '236', '237', '238', '239',
    '240', '241', '242', '243', '244', '245', '246', '247', '248', '249',
    '250', '251', '252', '253', '254', '255', '256', '257', '258', '259',
    '260', '261', '262', '263', '264', '265', '266', '267', '268', '269',
    '270', '271', '272', '273', '274', '275', '276', '277', '278', '279',
    '280', '281', '282', '283', '284', '285', '286', '287', '288', '289',
    '290', '291', '292', '293', '294', '295', '296', '297', '298', '299',
    '301', '302', '303', '304', '305', '306', '307', '308', '309',
    '310', '311', '312', '313', '314', '315', '316', '317', '318', '319',
    '320', '321', '322', '323', '324', '325', '326', '327', '328', '329',
    '330', '331', '332', '333', '334', '335', '336', '337', '338', '339',
    '340', '341', '342', '343', '344', '345', '346', '347', '348', '349',
    '350', '351', '352', '353', '354', '355', '356', '357', '358', '359',
    '360', '361', '362', '363', '364', '365', '366', '367', '368', '369',
    '370', '371', '372', '373', '374', '375', '376', '377', '378', '379',
    '380', '381', '382', '383', '384', '385', '386', '387', '388', '389',
    '390', '391', '392', '393', '394', '395', '396', '397', '398', '399',
    ]
```

```
tested_ragaraja_instructions = [
    '000', '001', '002', '003', '004', '005', '006', '007', '008', '009',
    '010', '011', '012', '013', '014', '015', '016', '017', '018', '019',
    '020', '021', '022', '023', '024', '025',
    '032', '033', '034', '035', '036', '037', '038', '039',
    '040', '041', '042', '043', '044', '045', '046', '047',
    '050', '051', '052', '053', '054', '055', '056', '057', '058', '059',
    '060', '061', '062', '063', '064', '065', '066', '067', '068', '069',
    '070', '071', '072', '073', '074', '075', '076', '077', '078', '079',
    '080', '081', '084', '085', '086', '087', '088', '089',
    '090', '091', '092', '093', '094', '095', '096', '097', '098', '099',
    '100', '101', '102', '103', '104', '105', '106', '107', '108', '109',
    '110', '111', '112', '113', '114', '115', '116', '117',
    '120', '121', '122', '123', '124', '125', '126', '127', '128',
    '140', '141', '142', '143', '144', '145', '146', '147',
    '150', '151', '152', '153', '154', '155', '156', '157', '158', '159',
    '160', '161', '162', '163', '164', '165', '166', '167', '168', '169',
    '170', '171',
    '189',
    '196', '197', '198',
    '201', '202', '203', '204', '205', '206', '207', '208', '209',
    '210', '211', '212', '213', '214', '215', '216', '217', '218', '219',
    '220', '221', '222', '223', '224', '225', '226', '227', '228', '229',
    '230', '231', '232', '233', '234', '235', '236', '237', '238', '239',
    '240', '241', '242', '243', '244', '245', '246', '247', '248', '249',
    '250', '251', '252', '253', '254', '255', '256', '257', '258', '259',
    '260', '261', '262', '263', '264', '265', '266', '267', '268', '269',
    '270', '271', '272', '273', '274', '275', '276', '277', '278', '279',
    '280', '281', '282', '283', '284', '285', '286', '287', '288', '289',
    '290', '291', '292', '293', '294', '295', '296', '297', '298', '299',
    '301', '302', '303', '304', '305', '306', '307', '308', '309',
    '310', '311', '312', '313', '314', '315', '316', '317', '318', '319',
    '320', '321', '322', '323', '324', '325', '326', '327', '328', '329',
    '330', '331', '332', '333', '334', '335', '336', '337', '338', '339',
    '340', '341', '342', '343', '344', '345', '346', '347', '348', '349',
    '350', '351', '352', '353', '354', '355', '356', '357', '358', '359',
    '360', '361', '362', '363', '364', '365', '366', '367', '368', '369',
    '370', '371', '372', '373', '374', '375', '376', '377', '378', '379',
    '380', '381', '382', '383', '384', '385', '386', '387', '388', '389',
    '390', '391', '392', '393', '394', '395', '396', '397', '398', '399',
    ]

nBF_instructions = ['000', '004', '008', '011', '020', '050', '051', '052',
                    '053', '054', '055', '056', '057', '058', '059', '060']

def source_filter(source, sfilter=ragaraja_v1):
    '''
    Checks a Ragaraja source code string and removes any instructions are
    not found in the filter

    @param source: Ragaraja source code string
    @type source: string
    @param sfilter: list of instructions allowed.
    Default = ragaraja_v1. Other defined lists are nBF_instructions (NucleotideBF)
    and tested_ragaraja_instructions.
    @return: Ragaraja source code string
    '''
    filtered_source = []
    spointer = 0
    while spointer < len(source):
        if source[spointer:spointer+3] in sfilter:
            filtered_source = filtered_source + [source[spointer:spointer+3]]
        spointer = spointer + 3
    return ''.join(filtered_source)

def nBF_to_Ragaraja(source):
    '''
    Converts NucleotideBF (nBF) source code to Ragaraja source code

    @param source: NucleotideBF (nBF) source code
    @type source: string
    @return: Ragaraja source code string
    '''
    converted = []
    for x in source:
        if x == 'G': converted.append('000')
        elif x == 'C': converted.append('004')
        elif x == 'A': converted.append('008')
        elif x == 'T': converted.append('011')
        elif x == '.': converted.append('020')
        elif x == 'R': converted.append('050')
        elif x == 'Y': converted.append('051')
        elif x == 'S': converted.append('052')
        elif x == 'W': converted.append('053')
        elif x == 'K': converted.append('054')
        elif x == 'M': converted.append('055')
        elif x == 'B': converted.append('056')
        elif x == 'D': converted.append('057')
        elif x == 'H': converted.append('058')
        elif x == 'V': converted.append('059')
        elif x == 'N': converted.append('060')
        else: converted.append('...')
    return ''.join(converted)

def activate_version(version=1):
    '''
    Function to only set tested instructions as usable.

    @param version: Define the version to activate. Default = 1. Allowable
    versions are
    - 0 (all currently tested instructions)
    - 0.1 (using NucleotideBF instructions)
    - 1 (as defined in Ling, MHT. 2012. An Artificial Life Simulation Library
    Based on Genetic Algorithm, 3-Character Genetic Code and Biological
    Hierarchy. The Python Papers.)
    '''
    instructions = None
    if version == 0.1: instructions = nBF_instructions
    if version == 1: instructions = ragaraja_v1
    if version == 0: instructions = tested_ragaraja_instructions
    for key in ragaraja.keys():
        if key not in instructions:
            ragaraja[key] = not_used
```

File name: t_ragaraja.py

```
'''
Testing Ragaraja Interpreter
Date created: 20th August 2012
Licence: Python Software Foundation License version 2

The tests are executed in sequence from test 1 to test N where the test N
specific code is added on the the concatenated codes/instructions from the
previous test in the form of

code(N) = code(N-1) + code(N)

Although it is possible to use unittest, it is probably mind-boggling to
try to understand long stretches of codes. It is probably easier to gradually
build up on the code and knowledge/state of the previous instructions. This
means that it is not possible to pass test N but fail test N-1.

Sample test case/scenario capsule -
```

```
1: {'restart': False
    'in_source': '008008008008',
    'forcedinarray': [0]*10
    'array': [4,0,0,0,0,0,0,0,0,0],
    'apointer': 0,
    'forcedindata': []
    'inputdata': [],
    'output': [],
    'out_source': '008008008008',
    'spointer': 12
   }

containing 10 options:
1. 'restart' - determines if the tester should clear concatenated instructions
    so far. Default = False, which means keep the clear concatenated
    instructions so far and add on the current instructions to execute.
    Not used unless to clean out concatenated instructions so far.
2. 'in_source' - the list of specific instructions to execute in this test,
    which will be appended to the list of concatenated instructions so far.
3. 'forcedinarray' - array/tape to be fed into register machine before code
    execution (code = concatenated instructions so far + in_source). If not
    defined, it will be set to [0,0,0,0,0,0,0,0,0,0].
4. 'array' - array/tape at the end of execution.
5. 'apointer' - array pointer at the end of execution.
6. 'forcedindata' - input data list to be fed into register machine before
    code execution (code = concatenated instructions so far + in_source).
    If not defined, it will be set to [].
7. 'inputdata' - input data list (defined by 'forcedindata') at the end of
    execution.
8. 'output' - output list at the end of execution.
9. 'out_source': concatenated instructions so far + in_source.
10. 'spointer': source pointer at the end of execution on 'out_source'.
'''

import sys
import os

import ragaraja as N

import register_machine as r

testdata = {
            # Testing 000, 004, 008, 011
            1: {'in_source': '008008008008',
                'array': [4,0,0,0,0,0,0,0,0,0],
                'apointer': 0,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008',
                'spointer': 12},
            2: {'in_source': '000000',
                'array': [4,0,0,0,0,0,0,0,0,0],
                'apointer': 2,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000',
                'spointer': 18},
            3: {'in_source': '011011011',
                'array': [4,0,-3,0,0,0,0,0,0,0],
                'apointer': 2,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000011011011',
                'spointer': 27},
            4: {'in_source': '004004004004',
                'array': [4,0,-3,0,0,0,0,0,0,0],
                'apointer': 8,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000011011011004004004004',
                'spointer': 39},
            5: {'in_source': '011011011011011',
                'array': [4,0,-3,0,0,0,0,0,-5,0],
                'apointer': 8,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000011011011004004004004011011011\
011011',
                'spointer': 54},
            6: {'in_source': '000000000',
                'array': [4,0,-3,0,0,0,0,0,-5,0],
                'apointer': 1,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000011011011004004004004011011011\
011011000000000',
                'spointer': 63},
            7: {'in_source': '008008008',
                'array': [4,3,-3,0,0,0,0,0,-5,0],
                'apointer': 1,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000011011011004004004004011011011\
011011000000000008008008',
                'spointer': 72},
            # Testing 001, 009
            8: {'in_source': '001009',
                'array': [4,3,-3,0,0,0,5,0,-5,0],
                'apointer': 6,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000011011011004004004004011011011\
011011000000000008008008001009',
                'spointer': 78},
            # Testing 002, 005
            9: {'in_source': '002005010',
                'array': [4,13,-3,0,0,0,5,0,-5,0],
                'apointer': 1,
                'inputdata': [],
                'output': [],
                'out_source': '008008008008000000011011011004004004004011011011\
011011000000000008008008001009002005010',
                'spointer': 87},
            # Testing 003, 043
            10: {'in_source': '043003',
                 'array': [4,13,-3,0,0,0,5,0,-5,0],
                 'apointer': 6,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
1011011000000000008008008001009002005010043003',
                 'spointer': 93},
            # Testing 006, 044
            11: {'in_source': '044006',
                 'array': [4,13,-3,0,0,0,5,0,-5,0],
                 'apointer': 9,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
1011011000000000008008008001009002005010043003044006',
                 'spointer': 99},
            # Testing 012, 013, 032, 033
            12: {'in_source': '012032013033',
                 'array': [4,13,-3,0,0,0,5,0,-5,-10],
                 'apointer': 9,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033',
                 'spointer': 111},
            # Testing 061, 062
            13: {'in_source': '004004004061004062',
                 'array': [4,13,-3,0,0,0,5,0,-5,-10],
                 'apointer': 6,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
062',
                 'spointer': 129},
            # Testing 016, 018
            14: {'in_source': '016018',
                 'array': [4,13,-3,0,0,0,5,0,-5,-10],
                 'apointer': 6,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
062016018',
                 'spointer': 135},
            # Testing 017, 019
            15: {'in_source': '017019',
                 'array': [4,13,-3,0,0,0,5,0,-5,-10],
                 'apointer': 6,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
062016018017019',
                 'spointer': 141},
            # Testing 046
            16: {'in_source': '046046',
                 'array': [4,13,-3,0,0,0,5,0,-5,-10],
                 'apointer': 6,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
062016018017019046046',
                 'spointer': 147},
            # Testing 081, 133
            17: {'in_source': '081133',
                 'array': [4,13,-3,0,0,0,0,-10,-5,5],
                 'apointer': 6,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
062016018017019046046081133',
                 'spointer': 153},
            # Testing 084, 085, 086
            18: {'in_source': '000084000085000086000097000098',
                 'array': [3.14159265358979323846,2.718281828459045,
                           -3,0,0,0,0,0,-1,1],
                 'apointer': 1,
                 'inputdata': [],
                 'output': [],
                 'out_source': '00800800800800000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
062016018017019046046081133000084000085000086000097000098',
                 'spointer': 183},
            # Testing 120, 121, 122
            19: {'in_source': '004120000121000122',
                 'array': [1,1,1,0,0,0,0,0,-1,1],
                 'apointer': 2,
                 'inputdata': [],
                 'output': [],
                 'out_source': '008008008008000000011011011004004004004011011011\
101101100000000000080080080010090020050100430030440060120320130330040040040061004\
062016018017019046046081133000084000085000086000097000098004120000121000122',
                 'spointer': 201},
            # Testing 065, 068, 071, 115
            20: {'in_source': '004004008004008071065068000008115',
                 'array': [6,1,1,0,0,0,0,0,-1,-4],
                 'apointer': 0,
                 'inputdata': [],
                 'output': [],
                 'out_source': '008008008008000000011011011004004004004011011011\
101101100000000000080080080010090020050100430030440060120320130330040040040061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
04008004008071065068000008115',
                 'spointer': 234},
            # Testing 021, 022, 038, 042, 047
            21: {'in_source': '021022047000038042',
                 'array': [6,0,1,0,0,0,0,0,-1,-4],
                 'apointer': 1,
                 'inputdata': [],
                 'output': [237],
                 'out_source': '008008008008000000011011011004004004004011011011\
101101100000000000080080080010090020050100430030440060120320130330040040040061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
04008004008071065068000008115021022047000038042',
                 'spointer': 252},
            # Testing 080, 116
            22: {'in_source': '009004116080',
                 'array': [7,5,1,0,0,0,0,0,-1,-4],
                 'apointer': 0,
                 'inputdata': [],
                 'output': [237],
                 'out_source': '008008008008000000011011011004004004004011011011\
101101100000000000080080080010090020050100430030440060120320130330040040040061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
04008004008071065068000008115021022047000038042009004116080',
                 'spointer': 264},
            # Testing 077, 087, 089, 088, 090, 094, 095, 096, 105, 108, 109, 117
            23: {'in_source': '077087009089088090094108096109117105095080',
                 'array': [8,5,1,0,0,0,0,0,-1,-4],
                 'apointer': 0,
                 'inputdata': [],
                 'output': [237],
                 'out_source': '008008008008000000011011011004004004004011011011\
101101100000000000080080080010090020050100430030440060120320130330040040040061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
04008004008071065068000008115021022047000038042009004116080077087009089088090009\
4108096109117105095080',
                 'spointer': 306},
            # Testing 091, 092, 093, 099, 100, 101, 102, 103, 104, 112, 113
            24: {'in_source': '000000091093099101092100102103104112113105080',
                 'array': [8,5,50,0,0,0,0,0,-1,-4],
                 'apointer': 2,
                 'inputdata': [],
                 'output': [237],
                 'out_source': '008008008008000000011011011004004004004011011011\
101101100000000000080080080010090020050100430030440060120320130330040040040061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
04008004008071065068000008115021022047000038042009004116080077087009089088090009\
4108096109117105095080000000091093099101092100102103104112113105080',
                 'spointer': 351},
            # Pushing [1,2,3,4,5,6,7,8,9] as input data into the test system
            25: {'in_source': '',
                 'array': [8,5,50,0,0,0,0,0,-1,-4],
                 'apointer': 2,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '00800800800800000001101101100400400400401101101\
1011011000000000000800800800100900200501004300304400601203201303300400400406100\4
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080',
                 'spointer': 351},
            # Testing 066, 067, 069, 070, 072, 073, 075, 076
            26: {'in_source': '066000067000069000070005000072073076075',
                 'array': [8,5,51,9,1,9,0,0,-1,-4],
                 'apointer': 1,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '00800800800800000001101101100400400400401101101\
1011011000000000000800800800100900200501004300304400601203201303300400400406100\4
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
069000070005000072073076075',
                 'spointer': 390},
            # Testing 140, 141, 142
            27: {'in_source': '140008141013142008',
                 'array': [8,5,41,9,2,9,1,0,-1,-4],
                 'apointer': 6,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '00800800800800000001101101100400400400401101101\
1011011000000000000800800800100900200501004300304400601203201303300400400406100\4
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
069000070005000072073076075140008141013142008',
                 'spointer': 408},
            # Testing 189, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
            # 301, 302, 303, 304, 305, 306, 307, 308, 309, 310
            28: {'in_source': '004005201000202000203000204000205000206000207000\
2080002090002100001893100003090003080003070003060003050003040003030003020003010\
00',
                 'array': [-4,-1,0,1,9,2,9,41,5,8],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '00800800800800000001101101100400400400401101101\
1011011000000000000800800800100900200501004300304400601203201303300400400406100\4
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000',
                 'spointer': 537},
            # Testing 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
            # 311, 312, 313, 314, 315, 316, 317, 318, 319, 320
            29: {'in_source': '211000212000213000214000215000216000217000218000\
219000220000320000319000318000317000316000315000314000313000312000311000',
                 'array': [8,5,41,9,2,9,1,0,-1,-4],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '008008008000000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
06201601801701904604608113300008400008500008600009700009800412000012100012200 40\
04008004008071065068000008115021022047000038042009004116080077087009089088 09009\
41080961091171050950800000000910930991010921001021031041121131050800660000 67000\
06900007000500007207307607514000814101314200800400520100020200020300020400 02050\
00206000207000208000209000210000189310000309000308000307000306000305000304000 30\
30003020003010002110002120002130002140002150002160002170002180002190002200003 20\
000319000318000317000316000315000314000313000312000311000',
                 'spointer': 657},
            # Testing 221, 222, 223, 224, 225, 226, 227, 228, 229, 230,
            # 321, 322, 323, 324, 325, 326, 327, 328, 329, 330
            30: {'in_source': '221000222000223000224000225000226000227000228000\
229000230000330000329000328000327000326000325000324000323000322000321000',
                 'array': [-4,-1,0,1,9,2,9,41,5,8],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '008008008000000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000211000212000213000214000215000216000217000218000219000220000320\
0003190003180003170003160003150003140003130003120003110002210002220002230002240\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000',
                 'spointer': 777},
            # Testing 231, 232, 233, 234, 235, 236, 237, 238, 239, 240,
            # 331, 332, 333, 334, 335, 336, 337, 338, 339, 340
            31: {'in_source': '231000232000233000234000235000236000237000238000\
239000240000340000339000338000337000336000335000334000333000332000331000',
                 'array': [8,5,41,9,2,9,1,0,-1,-4],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '008008008000000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000211000212000213000214000215000216000217000218000219000220000320\
0003190003180003170003160003150003140003130003120003110002210002220002230002240\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000231000232000233000234000235000236000237000238000239000240000\
340000339000338000337000336000335000334000333000332000331000',
                 'spointer': 897},
            # Testing 241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
            # 341, 342, 343, 344, 345, 346, 347, 348, 349, 350
            32: {'in_source': '241000242000243000244000245000246000247000248000\
249000250000350000349000348000347000346000345000344000343000342000341000',
                 'array': [-4,-1,0,1,9,2,9,41,5,8],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '008008008000000001101101100400400400401101101\
10110110000000000008008008001009002005010043003044006012032013033004004004061004\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000211000212000213000214000215000216000217000218000219000220000320\
0003190003180003170003160003150003140003130003120003110002210002220002230002240\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000231000232000233000234000235000236000237000238000239000240000\
3400003390003380003370003360003350003340003330003320003310002410002420002430002\
4400024500024600024700024800024900025000035000034900034800034700034600034500034\
4000343000342000341000',
                 'spointer': 1017},
            # Testing 251, 252, 253, 254, 255, 256, 257, 258, 259, 260,
            # 351, 352, 353, 354, 355, 356, 357, 358, 359, 360
            33: {'in_source': '251000252000253000254000255000256000257000258000\
259000260000360000359000358000357000356000355000354000353000352000351000',
                 'array': [8,5,41,9,2,9,1,0,-1,-4],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '008008008008000000001101101100400400400401101101\
1011011000000000000800800800100900200501004300304400601203201303300400400406100 4\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000211000212000213000214000215000216000217000218000219000220000320\
0003190003180003170003160003150003140003130003120003110002210002220002230002240\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000231000232000233000234000235000236000237000238000239000240000\
3400003390003380003370003360003350003340003330003320003310002410002420002430002\
4400024500024600024700024800024900025000035000034900034800034700034600034500034\
4000343000342000341000251000252000253000254000255000256000257000258000259000260\
000360000359000358000357000356000355000354000353000352000351000',
                 'spointer': 1137},
            # Testing 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,
            # 361, 362, 363, 364, 365, 366, 367, 368, 369, 370
            34: {'in_source': '261000262000263000264000265000266000267000268000\
269000270000370000369000368000367000366000365000364000363000362000361000',
                 'array': [-4,-1,0,1,9,2,9,41,5,8],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [237],
                 'out_source': '008008008008000000001101101100400400400401101101\
1011011000000000000800800800100900200501004300304400601203201303300400400406100 4\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000211000212000213000214000215000216000217000218000219000220000320\
0003190003180003170003160003150003140003130003120003110002210002220002230002240\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000231000232000233000234000235000236000237000238000239000240000\
3400003390003380003370003360003350003340003330003320003310002410002420002430002\
4400024500024600024700024800024900025000035000034900034800034700034600034500034\
4000343000342000341000251000252000253000254000255000256000257000258000259000260\
0003600003590003580003570003560003550003540003530003520003510002610002620002630\
0026400026500026600026700026800026900027000037000036900036800036700036600036500\
0364000363000362000361000',
                 'spointer': 1257},
            # Testing 271, 272, 273, 274, 275, 276, 277, 278, 279, 280,
            # 371, 372, 373, 374, 375, 376, 377, 378, 379, 380
```

```
          35: {'in_source': '271000272000273000274000275000276000277000278000\
279000280000380000379000378000377000376000375000374000373000372000371000',
               'array': [8,5,41,9,2,9,1,0,-1,-4],
               'apointer': 0,
               'forcedindata': [1,2,3,4,5,6,7,8,9],
               'inputdata': [1,2,3,4,5,6,7,8,9],
               'output': [237],
               'out_source': '008008008008000000011011011004004004004401101101\
101101100000000000080080080010090020050100430030440060120320130330040040040610 04\
062016018017019046046081133000084000085000086000097000098004120000121000122004 0\
040080040080710650680000081150210220470000380420090041160800770870090890880900 9\
410809610911710509508000000009109309910109210010210310411211310508006600006700 0\
069000070005000072073076075140008141013142008004005201000202000203000204000205 0\
002060002070002080002090002100001893100003090003080003070003060003050003040003 0\
300030200030100021100021200021300021400021500021600021700021800021900022000032 0\
000319000318000317000316000315000314000313000312000311000221000222000223000224 0\
002250002260002270002280002290002300003300003290003280003270003260003250003240 0\
032300032200032100023100023200023300023400023500023600023700023800023900024000 0\
340000339000338000337000336000335000334000333000332000331000241000242000243000 2\
440002450002460002470002480002490002500003500003490003480003470003460003450003 4\
400034300034200034100025100025200025300025400025500025600025700025800025900026 0\
000360000359000358000357000356000355000354000353000352000351000261000262000263 0\
002640002650002660002670002680002690002700003700003690003680003670003660003650 0\
036400036300036200036100027100027200027300027400027500027600027700027800027900 0\
280000380000379000378000377000376000375000374000373000372000371000',
               'spointer': 1377},
          # Testing 281, 282, 283, 284, 285, 286, 287, 288, 289, 290,
          # 381, 382, 383, 384, 385, 386, 387, 388, 389, 390
          36: {'in_source': '281000282000283000284000285000286000287000288000\
289000290000390000389000388000387000386000385000384000383000382000381000',
               'array': [-4,-1,0,1,9,2,9,41,5,8],
               'apointer': 0,
               'forcedindata': [1,2,3,4,5,6,7,8,9],
               'inputdata': [1,2,3,4,5,6,7,8,9],
               'output': [237],
               'out_source': '008008008008000000011011011004004004004401101101\
101101100000000000080080080010090020050100430030440060120320130330040040040610 04\
062016018017019046046081133000084000085000086000097000098004120000121000122004 0\
040080040080710650680000081150210220470000380420090041160800770870090890880900 9\
410809610911710509508000000009109309910109210010210310411211310508006600006700 0\
069000070005000072073076075140008141013142008004005201000202000203000204000205 0\
002060002070002080002090002100001893100003090003080003070003060003050003040003 0\
300030200030100021100021200021300021400021500021600021700021800021900022000032 0\
000319000318000317000316000315000314000313000312000311000221000222000223000224 0\
002250002260002270002280002290002300003300003290003280003270003260003250003240 0\
032300032200032100023100023200023300023400023500023600023700023800023900024000 0\
340000339000338000337000336000335000334000333000332000331000241000242000243000 2\
440002450002460002470002480002490002500003500003490003480003470003460003450003 4\
400034300034200034100025100025200025300025400025500025600025700025800025900026 0\
000360000359000358000357000356000355000354000353000352000351000261000262000263 0\
002640002650002660002670002680002690002700003700003690003680003670003660003650 0\
036400036300036200036100027100027200027300027400027500027600027700027800027900 0\
280000380000379000378000377000376000375000374000373000372000371000281000282000 2\
830002840002850002860002870002880002890002900003900003890003880003870003860003 8\
500038400038300038200038100 0',
               'spointer': 1497},
          # Testing 291, 292, 293, 294, 295, 296, 297, 298, 299,
          # 391, 392, 393, 394, 395, 396, 397, 398, 399
          37: {'in_source': '291000292000293000294000295000296000297000298000\
299000000399000398000397000396000395000394000393000392000391',
               'array': [5,41,9,2,9,1,0,-1,-4,8],
               'apointer': 8,
               'forcedindata': [1,2,3,4,5,6,7,8,9],
               'inputdata': [1,2,3,4,5,6,7,8,9],
               'output': [237],
               'out_source': '008008008008000000011011011004004004004401101101\
101101100000000000080080080010090020050100430030440060120320130330040040040610 04\
06201601801701904604608113300008400008500008600009700009800412000012100012200040\
04008004008071065068000008115021022047000038042009004116080077087009089088090 09\
```

```
062016018017019046046081133000084000085000086000097000098004120000121000122040\
040080040080710650680000081150210220470000380420090041160800770870090890880 9009\
```

```
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000211000212000213000214000215000216000217000218000219000220000320\
0003190003180003170003160003150003140003130003120003110002210002220002230002240\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000231000232000233000234000235000236000237000238000239000240000\
3400003390003380003370003360003350003340003330003320003310002410002420002430002\
4400024500024600024700024800024900025000035000034900034800034700034600034500034\
4000343000342000341000251000252000253000254000255000256000257000258000259000260\
0003600003590003580003570003560003550003540003530003520003510002610002620002630\
0026400026500026600026700026800026900027000037000036900036800036700036600036500\
0364000363000362000361000271000272000273000274000275000276000277000278000279000\
2800003800003790003780003770003760003750003740003730003720003710002810002820002\
8300028400028500028600028700028800028900029000039000038900038800038700038600038\
5000384000383000382000381000291000292000293000294000295000296000297000298000299\
000000399000398000397000396000395000394000393000392000391',
                'spointer': 1605},
           # Testing 007, 020, 023, 024, 025, 034, 035, 036, 045, 063, 064
           38: {'in_source': '0000070200230000000240010020030040050000025001002\
003004005006007008009010000034035034036045063000064',
                'array': [1,2,9,2,9,1,0,-1,-4,8],
                'apointer': 1,
                'forcedindata': [1,2,3,4,5,6,7,8,9],
                'inputdata': [2,3,4,5,6,7,8,9],
                'output': [237,-4,0],
                'out_source': '0080080080080000000110110110040040040040110110 1\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000231000232000233000234000235000236000237000238000239000240000\
3400003390003380003370003360003350003340003330003320003310002410002420002430002\
4400024500024600024700024800024900025000035000034900034800034700034600034500034\
4000343000342000341000251000252000253000254000255000256000257000258000259000260\
0003600003590003580003570003560003550003540003530003520003510002610002620002630\
0026400026500026600026700026800026900027000037000036900036800036700036600036500\
0364000363000362000361000271000272000273000274000275000276000277000278000279000\
2800003800003790003780003770003760003750003740003730003720003710002810002820002\
8300028400028500028600028700028800028900029000039000038900038800038700038600038\
5000384000383000382000381000291000292000293000294000295000296000297000298000299\
0000003990003980003970003960003950003940003930003920003910000070200230000000240\
0100200300400500002500100200300400500600700800901000003403503403604506300006400\
0000000144145',
                 'spointer': 1719},
            # Testing 146, 147, 156, 157, 165
            40: {'in_source': '146147157156165',
                 'array': [-0.2,-0.4,-1.8,-0.4,-1.6,-0.2,0,0.2,0.8,-1.6],
                 'apointer': 4,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [2,3,4,5,6,7,8,9],
                 'output': [237,-4,0],
                 'out_source': '008008008008000000001101101100400400400401101101\
1011011000000000000800800800100900200501004300304400601203201303300400400406100 4\
0620160180170190460460811330000840000850000860000970000980041200001210001220040\
0400800400807106506800000811502102204700003804200900411608007708700908908809009\
4108096109117105095080000000091093099101092100102103104112113105080066000067000\
0690000700050000720730760751400081410131420080040052010002020002030002040002050\
0020600020700020800020900021000018931000030900030800030700030600030500030400030\
3000302000301000211000212000213000214000215000216000217000218000219000220000320\
0003190003180003170003160003150003140003130003120003110002210002220002230002240\
0022500022600022700022800022900023000033000032900032800032700032600032500032400\
0323000322000321000231000232000233000234000235000236000237000238000239000240000\
3400003390003380003370003360003350003340003330003320003310002410002420002430002\
4400024500024600024700024800024900025000035000034900034800034700034600034500034\
4000343000342000341000251000252000253000254000255000256000257000258000259000260\
0003600003590003580003570003560003550003540003530003520003510002610002620002630\
0026400026500026600026700026800026900027000037000036900036800036700036600036500\
0364000363000362000361000271000272000273000274000275000276000277000278000279000\
2800003800003790003780003770003760003750003740003730003720003710002810002820002\
8300028400028500028600028700028800028900029000039000038900038800038700038600038\
5000384000383000382000381000291000292000293000294000295000296000297000298000299\
0000003990003980003970003960003950003940003930003920003910000070200230000000240\
0100200300400500002500100200300400500600700800901000003403503403604506300006400\
0000000144145146147157156165',
                 'spointer': 1734},
            # Testing 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 166,
            # 167, 168, 169, 170, 171
            41: {'restart': True,
                 'in_source': '020000020000020000020000020166171167170168169146\
147004148149000152154000151000153080150155',
                 'forcedinarray': [1,4,9,16,25,1,2,3,4,5],
                 'array': [0.5,2,4.5,14,4.95,1.75,38.2,1.5,2,2.5],
                 'apointer': 6,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [1,4,9,16,25],
                 'out_source': '020000020000020000020000020166171167170168169 14\
6147004148149000152154000151000153080150155',
                 'spointer': 90},
            # Testing 158, 159, 160, 196, 197, 198
            42: {'in_source': '197198196158159160080',
                 'forcedinarray': [1,4,9,16,25,1,2,3,4,5],
                 'array': [5,20,45,140,49.5,17.5,39,15,20,25],
                 'apointer': 6,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [1,4,9,16,25],
                 'out_source': '02000002000002000002000002016617116717016816914\
61470041481490001521540001510001530801501551971981961581591600 80',
                 'spointer': 108},
            # Testing 074, 078, 079
            43: {'in_source': '000000000000074078000079',
                 'forcedinarray': [1,4,9,16,25,1,2,3,4,5],
                 'array': [1,9,45,140,49.5,17.5,39,15,20,25],
                 'apointer': 1,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [1,4,9,16,25],
                 'out_source': '02000002000002000002000002016617116717016816914\
61470041481490001521540001510001530801501551971981961581591600800000000000000074\
078000079',
                 'spointer': 135},
            # Testing 037, 039, 040, 041, 110, 111, 114
            44: {'in_source': '037000039000040041110111080114',
                 'forcedinarray': [1,4,9,16,25,1,2,3,4,5],
                 'array': [1,25,1,24,49.5,17.5,39,15,20,25],
                 'apointer': 3,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [9,16],
                 'out_source': '02000002000002000002000002016617116717016816914\
61470041481490001521540001510001530801501551971981961581591600800000000000000074\
078000079037000039000040041110111080114',
                 'spointer': 165},
            # Testing 123, 124, 125, 126, 127, 128
            45: {'in_source': '123000124125000126000127000128',
                 'forcedinarray': [1,4,9,16,25,1,2,3,4,5],
                 'array': [1,25,1,1,0,1,0,0,20,25],
                 'apointer': 7,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [9,16],
                 'out_source': '02000002000002000002000002016617116717016816914\
61470041481490001521540001510001530801501551971981961581591600800000000000000074\
078000079037000039000040041110111080114123000124125000126000127000128',
                 'spointer': 195},
            # Testing 143, 161, 162, 163, 164
            46: {'in_source': '161000000163000000164005162004143',
                 'forcedinarray': [1,4,9,16,25,1,2,3,4,5],
                 'array': [1,0,1,0,20,25,0,1,25,1],
                 'apointer': 7,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [9,16],
                 'out_source': '02000002000002000002000002016617116717016816914\
61470041481490001521540001510001530801501551971981961581591600800000000000000074\
078000079037000039000040041110111080114123000124125000126000127000128161000000001\
63000000164005162004143',
                 'spointer': 228},
            # Testing 014, 015 (single loop)
            47: {'restart': True,
                 'in_source': '008008008008008008008008008014000008008008008\
008004020011015',
                 'forcedinarray': [0,0,0,0,0,0,0,0,0,0],
                 'array': [0,50,0,0,0,0,0,0,0,0],
                 'apointer': 0,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [10,9,8,7,6,5,4,3,2,1],
                 'out_source': '008008008008008008008008008014000008008008008\
008004020011015',
                 'spointer': 63},
            # Testing 014, 015 (double loop)
            48: {'restart': True,
                 'in_source': '0080080080080080140000080080080080140000080080080\
20004011015004011015001009',
                 'forcedinarray': [0,0,0,0,0,0,0,0,0,0],
                 'array': [0,0,60,0,0,5,0,0,0,0],
                 'apointer': 5,
                 'forcedindata': [1,2,3,4,5,6,7,8,9],
                 'inputdata': [1,2,3,4,5,6,7,8,9],
                 'output': [3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,51,54,
                            57,60],
                 'out_source': '0080080080080080140000080080080080140000008008008\
020004011015004011015001009',
                 'spointer': 75},
           }

tests = testdata.keys()
tests.sort()

full_source = ''
source = ''

def comparator(data, result):
    if data == result: return 'PASSED:'
    else: return 'FAILED:'

for t in tests:
    # ---------------------------
    # ------- PREPARE TEST -------
    # ---------------------------

    # Step 1: Check to concatenate source or restart source
    try:
        if testdata[t]['restart'] == True:
            full_source = full_source + source
            source = ''
    except KeyError: pass

    isource = source + testdata[t]['in_source']

    # Step 2: Get input data list from test (if any)
    try: inputdata = testdata[t]['forcedindata']
    except KeyError: inputdata = []

    # Step 3: get pre-execution tape (array) from test (if any)
    try: array = testdata[t]['forcedinarray']
    except KeyError: array = [0]*10

    # Step 4: Get expected results after execution
    oarray = testdata[t]['array']            # tape (array) after execution
    oapointer = testdata[t]['apointer']      # tape pointer
    oinputdata = testdata[t]['inputdata']    # input data list
    ooutput = testdata[t]['output']          # output list
    osource = testdata[t]['out_source']      # source
    ospointer = testdata[t]['spointer']      # source pointer

    # ---------------------------
    # ------- EXECUTE TEST -------
    # ---------------------------

    (array, apointer, inputdata, output, source, spointer) = \
        r.interpret(isource, N.ragaraja, 3, inputdata, array, 10)

    # -----------------------------------------------
    # ------- COMPARE EXPECTED RESULTS AFTER TEST -------
    # -----------------------------------------------

    print ' '.join(['Test number:', str(t),
                    ', Original source code:', str(isource)])
    print ' '.join(['    ', str(comparator(oarray, array)), 'array.',
                    'Expected array:', str(oarray),
                    'Actual array:', str(array)])
    print ' '.join(['    ', str(comparator(oapointer, apointer)),
                    'array pointer.',
                    'Expected array pointer:', str(oapointer),
                    'Actual array pointer:', str(apointer)])
    print ' '.join(['    ', str(comparator(oinputdata, inputdata)),
                    'input data list.',
                    'Expected input data list:', str(oinputdata),
                    'Actual input data list:', str(inputdata)])
    print ' '.join(['    ', str(comparator(ooutput, output)),
                    'output list.',
                    'Expected output list:', str(ooutput),
                    'Actual output list:', str(output)])
    print ' '.join(['    ', str(comparator(osource, source)),
                    'source code after execution.',
                    'Expected source:', str(osource),
                    'Actual source:', str(source)])
    print ' '.join(['    ', str(comparator(ospointer, spointer)),
                    'source code pointer after execution.',
                    'Expected source pointer:', str(ospointer),
                    'Actual source pointer:', str(spointer)])
    print '=========================================================='
# -----------------------------
# --------- REGISTERS ---------
# -----------------------------

print
print '===== Internal register state ====='
print 'Internal registers: ' + str(N.register)

# ---------------------------------------------
# --------- TESTING RANDOM OPERATIONS ---------
# ---------------------------------------------

print
print '===== Testing random operations ====='
print 'Testing 050 051 052 053 054 055 056 057 058 059 060'
print
random_source = '050051052053054055056057058059060'
random_source = random_source + random_source + random_source + \
                random_source + random_source + random_source
for x in range(20):
    tape = r.interpret(random_source, N.ragaraja, 3, [], [0]*10, 10)[0]
    print 'Test #' + str(x+1) + ', Tape: ' + str(tape)
print '===== End of random operations testing ====='

# ---------------------------
# --------- SUMMARY ---------
# ---------------------------

isource = full_source + isource + random_source
instruction_set = {}
for i in range(0, len(isource), 3): instruction_set[isource[i:i+3]] = ''
instruction_set = instruction_set.keys()
instruction_set.sort()
print
print 'Instruction set tested: '
print ' '.join(instruction_set)
print
print 'Number of instructions tested :' + str(len(instruction_set))
print
print '----- End of test -----'
```

4. References

Bersini, H. 2009. How artificial life relates to theoretical biology. Origins of Life: Self-Organization and/or Biological Evolution?, 61-78.

Harvey, I. 2011. The microbial genetic algorithm. Lecture Notes in Computer Science 5778, 126-133.

Kim, KJ, Cho, S.B. 2006. A comprehensive overview of the applications of artificial life. Artificial Life 12, 153-182.

Langton, CG. 1986. Studying artificial life with cellular automata. Physica D: Nonlinear Phenomena 22, 120-149.

Lim, JZR, Aw, ZQ, Goh, DJW, How, JA, Low, SXZ, Loo, BZL, Ling, MHT. 2010. A genetic algorithm framework grounded in biology. The Python Papers Source Codes 2: 6.

Ling, MHT. 2012. An artificial life simulation library based on genetic algorithm, 3-character genetic code and biological hierarchy. The Python Papers 7: 5.

Ward, MP, Laffan, SW., and Highfield, LD. 2011. Disease spread models in wild and feral animal populations: application of artificial life models. Revue Scientifique et Technique 30, 437-446.

Watson, R. 2012. Is evolution by natural selection the algorithm of biological evolution? In C. Adami, DM. Bryson, C. Ofria, and RT. Pennock (eds.) Artificial Life XIII: Proceedings of the Thirteenth International Conference on the Synthesis and Simulation of Living Systems Artificial Life XIII: 13th International Conference on the Simulation and Synthesis of Living Systems, pp. 121-128.