Assignment 1

Name: Zhijun Wei
Student Number: 21027550
Email: z29wei@uwaterloo.ca

Question 1:

(a) A test case that does not execute the fault:
a = 1, b = 1
The input is invalid, so the program would raise an exception and quit before line 8.

(b) A test case that executes the fault, but does not cause an error:
a = [[1, 2], [3, 4]], b = [[5],[6],[7]]
In this case, the program executes line 8, which is the fault, but will not use the result of line 8 to do the further calculation, since the condition in line 9 is not satisfied. Therefore it will not cause an error and will raise "incompatible dimensions" which is expected.

(c) A test case that results in an error, but not in a failure:
a = [[1, 1], [1, 1]], b = [[2, 2], [2, 2]]
In this case, the a and b are both 2*2 matrices. The fault is executed, and the result of line 8 is used to do the further calculation, which will result in an error. However, because len(b) and len(b[0]) are both equal to 2, the program will not calculate a wrong final result, therefore, it will not result in a failure.

(d) States:
State 0:
a = [[5, 7], [8, 21]]
b = [[8], [4]]
n = undefined
p = undefined
q = undefined
p1 = undefined
c = undefined
i = undefined
j = undefined
pc = matmul(...)

State 1:
a = [[5, 7], [8, 21]]
b = [[8], [4]]
n = 2
p = 2

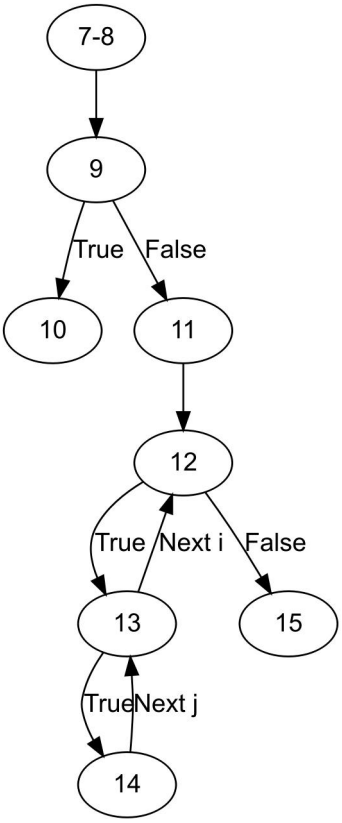
q = undefined
p1 = undefined
c = undefined
i = undefined
j = undefined
pc = n, p = len(a), len(a[0])

State 2:
a = [[5, 7], [8, 21]]
b = [[8], [4]]
n = 2
p = 2
q = 2
p1 = 1
c = undefined
i = undefined
j = undefined
pc = q, p1 = len(b), len(b[0])

State 3:
a = [[5, 7], [8, 21]]
b = [[8], [4]]
n = 2
p = 2
q = 2
p1 = 1
c = undefined
i = undefined
j = undefined
pc = if p != p1

State 4: the first error state
a = [[5, 7], [8, 21]]
b = [[8], [4]]
n = 2
p = 2
q = 2
p1 = 1
c = undefined
i = undefined
j = undefined
pc = raise ValueError("Incompatible dimensions")

(e)



Question 2:

(a)

```
class RepeatUntilStmt(Stmt):
    """A repeat-until statement"""
    def __init__(self, stmt, cond):
        self.stmt = stmt    # The statement S to be repeatedly executed
        self.cond = cond    # The Boolean expression b to be evaluated
```

(b)

Case 1: b evaluates to true after executing S once

$\langle S, q \rangle \Downarrow q' \quad \langle b, q' \rangle \Downarrow \text{true}$

$\langle \text{repeat } S \text{ until } b, q \rangle \Downarrow q'$

Case 2: b evaluates to false after executing S

$\langle S, q \rangle \Downarrow q'' \quad \langle b, q'' \rangle \Downarrow \text{false} \quad \langle \text{repeat } S \text{ until } b, q'' \rangle \Downarrow q'$

$\langle \text{repeat } S \text{ until } b, q \rangle \Downarrow q'$

(c)

<x := x - 1, [x := 1]> ⇓ [x := 0] <x <= 0, [x := 0]> ⇓ true

<x := x - 1, [x := 2]> ⇓ [x := 1] <x <= 0, [x := 1]> ⇓ false <repeat x := x - 1 until x <= 0; [x := 1]> ⇓ [x := 0]

<x := 2, [ ]> ⇓ [x := 2] <repeat x := x - 1 until x <= 0; [x := 2]> ⇓ [x := 0]

<x := 2 ; repeat x := x - 1 until x <= 0; [ ]> ⇓ [x := 0]

(d)
For “repeat S until b”, now we already have:

Case 1: b evaluates to true after executing S once

< S, q > ⇓ q' < b, q' > ⇓ true

< repeat S until b, q > ⇓ q'

Case 2: b evaluates to false after executing S

< S, q > ⇓ q'' < b, q'' > ⇓ false < repeat S until b, q'' > ⇓ q'

< repeat S until b, q > ⇓ q'

Lemma 2.5 from "Semantics with Applications: An Appetizer" generally involves demonstrating that two different constructs in the language yield the same resultant state when executed from the same initial state.

So we make the derivation of state “S; if b then skip else(repeat S until b)”, and prove that it can yield the same resultant as “repeat S until b”.

Case 1: b evaluates to true after executing S once

<skip, q'> ⇓ q'

< S, q > ⇓ q' < b, q' > ⇓ true <if b then skip else(repeat S until b), q'> ⇓ q'

< S; if b then skip else(repeat S until b), q > ⇓ q'

The the derivation is same as the derivation of “repeat S until b” when b is true.

Case 2: b evaluates to false after executing S

< repeat S until b, q'' > ⇓ q'

< S, q > ⇓ q'' < b, q'' > ⇓ false <if b then skip else(repeat S until b), q''> ⇓ q'

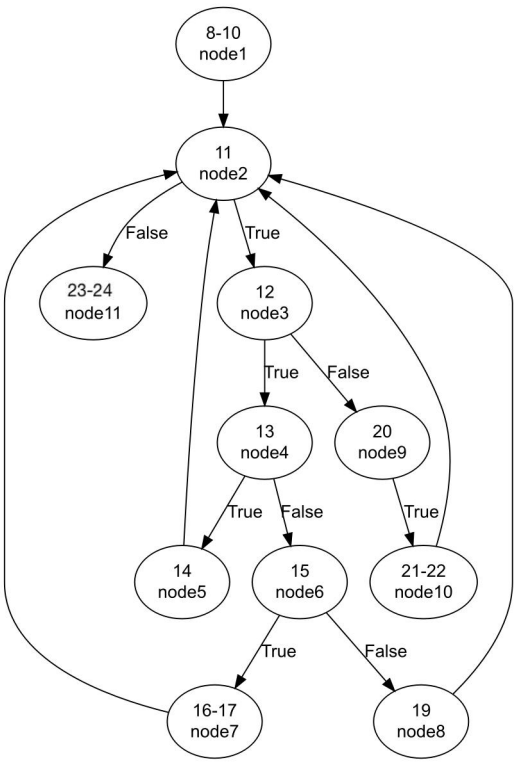
< S; if b then skip else(repeat S until b), q > ⇓ q'

The the derivation is same as the derivation of “repeat S until b” when b is false.

Therefore, “repeat S until b” is semantically equivalent to “S; if b then skip else(repeat S until b)”

Question 3:

(a)



(b)
TR_{NC}={node1, node2, node3, node4, node5, node6, node7, node8, node9, node10, node11}

TR_{EC}={(node1, node2), (node2, node3), (node3, node4), (node4, node5), (node4, node6), (node6, node7), (node6, node8), (node3, node9), (node9, node10), (node2, node11), (node5, node2), (node7, node2), (node8, node2), (node10, node2)}

TR_{EPC}={(node1, node2, node3), (node2, node3, node4), (node2, node3, node9), (node3, node4, node5), (node3, node4, node6), (node3, node9, node10), (node4, node5, node2), (node4, node6, node7), (node4, node6, node8), (node5, node2, node3), (node5, node2, node11), (node6, node7, node2), (node6, node8, node2), (node7, node2, node3), (node7, node2, node11), (node8, node2, node3), (node8, node2, node11), (node9, node10, node2), (node10, node2, node3), (node10, node2, node11)}

Question 4:

(a) The line that was not covered:
int.py: 179-197
These lines include the main function and the function that is only called in the main function. These functions can not be covered by test and do not need to be covered, since the main function is the entry point of the application, and can only be visited inside the module.

parser.py: 481-482
These two lines are the function _NEWLINE_. The newline is not a valid syntax since the parser cannot start a new line itself, so it is impossible to cover.

parser.py: 590-609
The reason is the same as int.py: 179-197. These lines include the main function. The main function can not be covered by the test and does not need to be covered, since it is the entry point of the application, and can only be visited inside the module.

(b) The line that was not covered in addition to part(a):
int.py: 90, 111
The input of the program must have valid operators. So if we want to cover these two lines, it will raise an error. Thus these two lines are hard to cover.

Conclusion:
Overall, the test cases can cover most of the branches, with the percentage of 97%. So we can conclude that the interpreter is correct and likely to work. The branch coverage is only missing two lines of code compared to statement coverage which means the testing is quite thorough, covering most possible code execution branches, and the interpreter is able to raise an error if the input format is invalid. Additionally, please note that when running the test, there will be some errors, because I want to cover more codes that related to raising the error when doing the wrong assertion and assumption.