A zero mark will be given if you copy someone else's work or you let someone copy your work.

COMP122/22-06 Data Structures and Algorithms	0 – 35 points
Stacks and Queues	2022-02-10 *Due Date* — 2022-02-17
Class Code	
Student No.	DO NOT WRITE YOUR NAME

† To test your code, you may put the following at the start of you code.

```
from linked_list import LnLs as Stack
from linked_list_queue import LQueue as Queue
```

1. Write a function to reverse the order of elements on stack s, by using two additional stacks a and b (initially empty), without any other variables. For example,

   (Top) 1,2,3,4,5 (Bottom) ⇒ (Top) 5,4,3,2,1 (Bottom) **6 points**

```
def reverse(s, a, b):

    while s:                    1
        a.push(s.pop())         1

    while a:                    1
        b.push(a.pop())         1

    while b:                    1
        s.push(b.pop())         1
```

   (1).

2. Write a function to duplicate the elements of stack s on top the original elements, by using two additional stacks a and b (initially empty), without any other variables. For example,

   (Top) 1,2,3,4,5 (Bottom) ⇒ (Top) 1,2,3,4,5,1,2,3,4,5 (Bottom) **7 points**

```
def dup_ss(s, a, b):

    while s:                    1
        a.push(s.pop())         1

    while a:                    1
        s.push(a.top())         1
        b.push(a.pop())         1

    while b:                    1
        a.push(b.pop())         1

    while a:                    1
        s.push(a.pop())         1
```

   (2).

3. Write a function to duplicate the elements of stack s on top the original elements, by using only one additional queue q (initially empty), and one integer variable m. **11 points**

```
def dup_sq(s, q):
```

```
    m = 0                          (1)
    while s:                       (1)
        q.push_back(s.pop())
        m += 1                     (2)

    while n > 0:                   (1)
        s.push(q.top())
        q.push_back(q.pop())
        m -= 1                     (3)

    while q:
        s.push(q.pop())            (1)

    while s:
        q.push_back(s.pop())       (1)

    while q:
        s.push(q.pop())            (1)
```

(3)

4. A postfix string consists of operators (`+`, `-`) and single letter variables (`t`, `u`, `v`, `w`, `x`, `y`, `z`), with an operator following two operands, each operand is either a variable or another postfix string, for example, '`wx+yz+-`'. Postfix strings can represent infix expressions without the need of parentheses. The example postfix string represents '`((w+x)-(y+z))`'. A postfix string p can be translated to the equivalent infix expression with the help of a stack s. We walk through the postfix string, for each character c,

   - if c is a variable, we push it to the stack,
   - if c is an operator, we pop two strings from the stack as q and r, and push the string `'('+r+c+q+')'` back to the stack.

   Finally, there remains one string on the stack, that is the result. Complete this function. **11 points**

```
def postfix_to_infix(p, s):
```

```
    for c in p:                    (2)
        if c in 'tuvwxyz':         (2)
            s.push(c)              (1)
        elif c in '+-':            (2)
            q = s.pop()            (1)
            r = s.pop()            (1)
            s.push('('+r+c+q+')')  (1)

    return s.pop()                 (1)
```

(4)

Write out the postfix string for '`t-((u+v)-((w+x)+(y-z)))`', and use it to test your function.