fullName: _____ andrewID: _____ section: ____

15-112 F26 Quiz3 Version A
Time: 20 Minutes

You must write your name on this paper and hand this back in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. Do not unstaple any pages. All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your quiz. We will not grade anything on scrap paper.

You may not ask questions during the quiz, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the quiz, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use material we have not yet covered. Thus, do not use lists, tuples, sets, dictionaries, OOP, or recursion. Also: do not use eval() or exec().

Also: If a method such as split() or splitlines() creates a list, you may only loop over the result of that method. You may not store the result in a variable or explicitly index into the result.

For all problems, assume that the required imports (such as 'import math') are present.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than the specified time noted above on this assessment.

Code Tracing [20 pts, from Comp3]

Fill in the Blank [20 pts, 10 pts each]

FitB1: [10 pts]

Fill in the blank so f(s) == g(s) for all strings s.

```
def f(s):
    if not s.isalpha():
        return ''
    r = ''
    for c in s:
        if 'A' <= c <= 'Z':
            offset = ord(c) - ord('A')
            r += chr(ord('a') + offset)
        else:
            r += c
    return r

def g(s):
    if not s.isalpha():
        return ''

    return ____________________________________________________________
```

FitB2: [10 pts]

Fill in the blank so f(s) == g(s) for all strings s.

```
def f(s):
    if not s.isalpha():
        return ''
    while 'AA' in s:
        s = s.replace('AA', 'A')
    return s

def g(s):
    if not s.isalpha():
        return ''
    r = ''
    for c in s:
        if (c != 'A') or (r == '') or (_____________________________):
            r += c
    return r
```

FR1: evalSum [30 pts]

Note: you may not use eval() or exec() or any similar function that trivializes this problem.

Background: we will say that s is a valid sum (a coined term) if:

- s is a string, and
- s contains only non-negative integers separated by + signs (so, for example, s does not contain spaces), and
- s contains at least one + sign.

With that, write the function evalSum(s) that takes an arbitrary value s (which may or may not be a string) and returns the sum of the integers in s if s is a valid sum, or None otherwise.

Here are some test cases:

```
assert(evalSum('1+2') == 3)
assert(evalSum('1+20+300') == 321)
assert(evalSum('42') == None)    # no + sign
assert(evalSum(42) == None)      # 42 is not a string
assert(evalSum('1+-2') == None)  # '-2' contains the non-digit '-'
assert(evalSum('1+2+') == None)  # no integer after trailing '+'
assert(evalSum('1 + 2') == None) # spaces are non-digits
assert(evalSum('1++2') == None)  # consecutive + signs
```

Write your solution on the next page.

FR2: fixAnagram [30 pts]

Background:

Recall that two strings s and t are anagrams if they are formed by exactly the same letters (perhaps in a different order).

Also, we will say that two strings s and t are near-anagrams (a coined term) if:

- s and t are both non-empty strings, and
- s and t are the same length but not equal, and
- s and t both contain only uppercase letters, and
- no letter occurs twice in either s or t, and
- you can change exactly one letter in t to make t an anagram of s.

With that, write the function fixAnagram(s, t) that takes two string s and t that are near anagrams (this is guaranteed so you do not have to check for it), and returns the anagram of s formed by changing exactly one letter in t.

For example, consider fixAnagram('ABC', 'DAB'). Here, we see that if we change the 'D' in t to 'C', we get 'CAB' which is an anagram of 'ABC'. Thus:

```
assert(fixAnagram('ABC', 'DAB') == 'CAB')
```

Here are some more test cases:

```
assert(fixAnagram('X', 'Y') == 'X')

assert(fixAnagram('WXYZ', 'ZYAW') == 'ZYXW')
```

Write your solution on the next page.

BonusCT

These CTs are optional, and intended to be very challenging. They are worth very few points. Indicate what the following code prints. Place your answers (and nothing else) in the boxes below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1: [1pt]

```
def bonusCt1(s):
    r = 0
    for c in s:
        if c.isalpha():
            d = math.ceil((ord(c) - ord('A')) / 25)
            if 0 <= d <= 1: r += d
        else:
            r *= 10
    return r
print(bonusCt1('This IS Awesome!'))
```

bonusCT2: [1pt]

```
def bonusCt2(s):
    t = 0
    for i in range(len(s)):
        for j in range(i+1, len(s)+1):
            try:
                t += float(s[i:j])
            except:
                pass
    return round(t, 1) # rounds to nearest 10th

print(bonusCt2('1*2.3==2.3'))
```