Last Name: _____ First Name: _____ Cornell NetID, all caps: _____

Circle your lab: Tu 12:20 Tu 1:25 Tu 2:30 Tu 3:35 W 12:20 W 1:25 W 2:30 W 3:35

CS 1110 Prelim 2 April 16th, 2013

This 90-minute exam has **??** questions worth a total of **??** points. When permitted to begin, scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

If a question does not explicitly ask for you to write an invariant, you don't have to for that problem. However, we strongly recommend that you provide comments explaining the meaning of your variables if you think they might be unclear to the graders.

The second page of this exam gives you the specifications for some useful functions.

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help.
We also ask that you not discuss this exam with students who are scheduled to take a later makeup.
Academic Integrity is expected of all students of Cornell University at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use or receive unauthorized aid in this examination.

Signature: _____ Date _____

The Important First Question:

1. [2 points] When allowed to begin, write your last name, first name, and Cornell NetID at the top of each page, and circle your lab time on the top of this page.

Last Name: _____ First Name: _____ Cornell NetID: _____

For reference:

`lt.index(item)`	Returns: index of first occurrence of `item` in list `lt`; raises an error if `item` is not found.
`range(n)`	Returns: the list `[0, 1, 2, ..., n-1]`
`lt[i:j]`	Returns: A new list `[lt[i], lt[i+1], ..., lt[j-1]]` under ordinary circumstances. Returns `[]` if `i` $\geq$ `len(lt)`.

Run LaTeX again to produce the table

2. [9 points] **Recursion**. In the spirit of card games, this question is about shuffling cards—or more generally, shuffling lists. When you shuffle cards you make two stacks, then interleave them, and if you were to do it perfectly, the even numbered cards would come from one stack and the odd numbered cards from the other stack. We call this even-odd interleaving a "perfect shuffle." Of course, you can do the same thing with any two lists, regardless of what type they contain.

   Your job is to write a recursive function that shuffles together two lists in this way. Here is the specification:

```
def shuffle(a, b):
    """Perfectly shuffle two decks.  Return a new list that contains the items
    in a and b, interleaved in the order a[0], b[0], a[1], b[1], a[2], ....
    If one list is longer than the other, the extra items go at the end.
    """
```

   Solution:

```
    if len(a) == 0 or len(b) == 0:
        # [+1 and +1]
        return a+b
        # [+2 (for correct handling of base case(s). It's wrong to
        #  return a or b (that would not be a new list)]
    else:
        return [a[0], b[0]] + shuffle(a[1:],b[1:])
        # +1: pulling first item from each list
        # +2: putting the first items at beginning of list,
        #     in right order
        # +1: calling shuffle
        # +1: ... on the right things
```

   Write the code for your recursive implementation in the space above. Here are some examples to illustrate further:

a	b	shuffle(a,b)
$[1, 2, 3]$	$[4, 5, 6]$	$[1, 4, 2, 5, 3, 6]$
$[1, 2, 3, 4, 5]$	$[10, 20, 30]$	$[1, 10, 2, 20, 3, 30, 4, 5]$
$[1, 2, 3]$	$[10, 20, 30, 40, 50]$	$[1, 10, 2, 20, 3, 30, 40, 50]$
$[]$	$[1, 3, 5]$	$[1, 3, 5]$

3. [8 points] **Broken invariant**.

Your friends Alice and Ben gave you the following functions, which have the same specification and are annotated with loop invariants and postconditions.

```
def even_first_A(x):
    """Organize the list <x> so that even numbers precede odd numbers.
    Pre: x is a list of integers."""
    i = 0
    j = len(x)-1
    # Inv: x[..i-1] are all even; x[j+1..] are all odd
    while j != i+1:
        if x[i] % 2 == 0:
            i = i + 1
        else:
            x[i],x[j] = x[j],x[i]
            j = j - 1
    # Post: x[..i-1] are all even; x[i..] are all odd

def even_first_B(x):
    """Organize the list <x> so that even numbers precede odd numbers.
    Pre: x is a list of integers."""
    i = -1
    j = len(x)
    # Inv: x[..i] are all even; x[j..] are all odd
    while j != i+1:
        if x[i+1] % 2 == 0:
            i = i + 1
        else:
            x[i],x[j] = x[j],x[i]
            j = j - 1
    # Post: x[..i] are all even; x[i+1..] are all odd
```

Alice and Ben both claim their code is correct because they implemented using invariants, and they are sure the invariants are correct. Indeed, their invariants and postconditions are fine; however, unit testing reveals that in fact neither of these functions works correctly.

Your job: correct each of these two functions so that it agrees with the written invariant and postcondition, and thereby works correctly. In each case, do so by changing **exactly one line**. For each function, show the fix by crossing out the offending line and then writing the correct version next to it.

Hint: if you're stuck, try implementing these functions from the invariant yourself on the back side of the previous page, and then compare your answer with Alice and Ben's. (But we will only grade corrections to the code above.)

Solution: A: wrong stopping condition (i < j+1, or i != j + 1) [2 for line, 2 for right answer]. Note that i<j fails to guarantee the postcondition; i <= j+1 fails on the empty list. B: wrong swap (x[i+1], x[j-1]). [same points, 4 total]

4. [9 points] The motivation for this problem is analyzing patterns in a sequence. Finish the implementation of the following function; note that we have started it off for you. Your implementation must use a loop to determine the length of the first run of zeroes in the list `data`, and then use recursion to determine the final answer.

```
def max_zrun_length(data):
    """Returns: length of the longest run of zeroes in data, which is a
    list of ints (possibly empty).

    Examples:   max_zrun_length([0,0,1,0,0,0]) is 3
                max_zrun_length([2,0,4,0,5]) is 1
                max_zrun_length([3,-2]) is 0
                max_zrun_length([]) is 0
    """

    if 0 not in data:
        return 0

    # now data is guaranteed to contain at least one 0
    i = data.index(0)  # i is the position of the first 0
```

Solution:

```
    # Inv: data[i..j-1] is known to be 0
    j = i + 1
    # [+1]

    while j < len(data) and data[j] == 0:
    # [+1 for checking len]
    # [+1 for checking len FIRST]
    # [+1 for non-zeroness for termination. -1 for return in loop]
       j += 1
       # [ +1 increment]

    firstlength = j - i  # length of first run of zeroes.
    # [+1]

    return max(firstlength, max_zrun_length(data[j:]))
    # data[j:] is empty if j == len(data)
    # [+1 max, +1 recursive call, +1 on right thing.
    # -1 for two recursive calls]

########   Alternate loop: ###############
    rlength = 1 # length of run seen so far; i is  index of last zero seen
    while i+1 < len(data) and data[i+1] == 0:
       rlength += 1; i=i+1
```

5. (a) [7 points] Write the body of the `__init__` method for the class whose spec is given below.

```
class User(object):
    """An instance is a user in a social network.

    Instance variables:
    username [string]: The name this user will use.  Can be any string but ''.
    bf [User]: The best friend of this user (None if they have no best friend).
               Best-friendship is mutual: if a is b's best friend, then
               b must be a's best friend, too.
    """

    def __init__(self, username, bf):
        """Initializer: a new user with username <username>. *If* <bf>
        is not None and <bf> does not already have a best friend, then
        this user's bf is <bf>, and <bf>'s best friend is this user.  But
        if <bf> is None or <bf> already has a best friend, then this
        user's bf should be None instead.

        So, after the sequence of statements
            u1 = User('1', None)
            u2 = User('2', u1)
            u3 = User('whee', u2)
        u1 should be u2's best friend and u2 should be u1's best friend.
        u3 should have no best friend, because u2 is already 'taken'.

        Pre: <username>: nonempty string. <bf>: either another user or None.
        """
```

Solution:

```
        self.username = username # [+1]
        if bf is not None and bf.bf is None: #[+1 for 1st check, +1 for
                                             #  2nd check, +1 for order]
            bf.bf = self  # [+1]
            self.bf = bf # [+1]
        else:
            self.bf = None # [+1]
```

(b) [4 points] Write the body of the `breakup` function whose spec is given below.

```
def breakup(users):
    """Make every user in <users> have no best friend (and hence their former
    best friends should have no best friend either).   Note that some of the
    items in <users> may already have no best friend.

    Precondition: <users> is a list of users. (May be empty.)
    """
```

Solution:

```
    for u in users:  # [+1]
        # have to be careful about the order this is done in
        if not u.bf is None: # [+1]
            u.bf.bf = None # [+1]
            u.bf = None # [+1]
```

Did you write your name and netID on each page, circle your lab time on the front, and re-read all specs?