Last Name: _____ First: _____ Netid: _____

CS 1110 Final, December 7th, 2017

This 150-minute exam has 8 questions worth a total of 100 points. 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.

It is a violation of the Academic Integrity Code to look at any exam other than your own, look at any reference material, or otherwise give or receive unauthorized help.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```
def foo():
    if something:
        do something
        do more things
    do something last
```

Unless you are explicitly directed otherwise, you may use anything you have learned in this course.

Question	Points	Score
1	2	
2	13	
3	14	
4	12	
5	14	
6	14	
7	17	
8	14	
Total:	100	

The Important First Question:

1. [2 points] Write your last name, first name, and netid at the top of each page.

Throughout this exam, there are several questions on sequences (strings and lists). All sequences support slicing. In addition, you may find the following sequence expressions below useful (though not all of them are necessary).

Expression	Description
`len(s)`	**Returns**: number of elements in sequence `s`; it can be 0.
`x in s`	**Returns**: `True` if `x` is an element of sequence `s`; `False` otherwise.
`s.index(x)`	**Returns**: index of the FIRST occurrence of `x` in `s`. Raises a `ValueError` if `x` is not found.
`x.append(a)`	(**Lists Only**) Adds `a` to the end of list `x`, increasing length by 1.
`x.remove(a)`	(**Lists Only**) Removes first occurrence of `a` in `x`, decreasing length by 1.
`x.extend(y)`	(**Lists Only**) Appends each element in `t` to the end of list `x`, in order.
`x.insert(i,y)`	(**Lists Only**) Inserts `y` at position `i` in list `x`. Elements after position `i` are shifted to the right.

2. [13 points total] **Short Answer**

(a) [4 points] Consider the following assignment statements.

```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
>>> b[0] = [10,11]
>>> b[1][0] = 99
```

What are the values `a` and `b` after all these assignments? Explain your answer.

`a` is `[[1,2,3],[4,5,6],[99,8,9]]`

`b` is `[[10,11],[99,8,9]]`

The second line (the slice) puts `[[4,5,6],[99,8,9]]`. The third line replaces the first element of this 2D list with another list. The last line goes inside of the list `[7,8,9]` and changes the first element. Because of how slicing works, the folder for this list is shared by both `a` and `b`, so changes to one affect another.

(b) [2 points] What is the difference between a function and a method?

A *method* is a function defined inside of the body of a class. Its name is stored in the class folder, and not inside a module or global space. A method is called differently than a function, with the first argument before the method name, rather than inside of the parentheses.

(c) [3 points] What is an assertion? How is it related to an `assert` statement?

An assertion is a property or statement (about code) that is either True or False. Assertions include preconditions, post conditions, and invariants. An `assert` statement is a way in Python of enforcing an assertion.

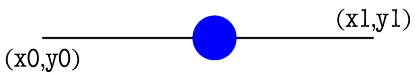
(d) [4 points] Describe the four steps that happen when you call a constructor.

When called, the constructor does the following:

- It creates a new object (folder) of the class.
- It puts the folder into heap space
- It executes the method `__init__` defined in the body of the class. In doing so, it
  - Passes the folder name to that parameter `self`
  - Passes the other arguments in order
  - Executes the commands in the body of `__init__`
- When done with `__init__` it returns the object (folder) name as final value of the expression (e.g. the constructor call).

3. [14 points] **Classes and Subclasses**

For this question, you are going to use the classes of Assignment 7 to make a `GSlider`. Shown to the right, this is a GUI element that you used to change the colors in Assignment 3. A slider is built up of two graphical objects, a `GPath` and a `GEllipse`. The former is the line that displays the path of the slider. The latter is the knob that the user drags to control the slider.



One easy way to make a GUI element composed of two different objects is to make the class a subclass on one and have the other as an attribute. That is what we have done on the next page. `GSlider` is a subclass of `GPath` but it has a `_knob` attribute for the `GEllipse`.

For this question, you only need to pay attention to the `points` attribute of `GPath`; all other attributes can be left to their default values. The `points` attribute is an even list of floats expressing the points the path goes through. For example, a line segment from (0,1) to (5,-3) would have `points` attribute [0,1,5,-3].

For the class `GEllipse` you only need to know the following attributes.

Attribute	Invariant	Description
`x`	`float`	x-coordinate of the ellipse center.
`y`	`float`	y-coordinate of the ellipse center.
`width`	`float > 0`	The width along the central horizontal axis.
`height`	`float > 0`	The height along the central vertical axis.
`fillcolor`	`str`	The interior color (represented as the name, e.g. `'blue'`).

Implemeting mouse control is too messy for an exam, so we control the slider with an additional attribute called `_value`. This attribute must be in sync with the knob. When it changes, the knob moves, and vice versa. This expressed by the specification on the next page.

With this in mind, implement this class on the next page. We have provided the specifications for the methods `__init__` and `draw`. You should fill in the missing details to meet these specifications. In addition, you must add the getters and setters (where appropriate) for the new attributes. Remember that setters must have preconditions to enforce the attribute invariants.

Hint: The attributes in `GPath` and `GEllipse` work like they do in Assignment 7, and have invisible setters and getters. Therefore, you never have to enforce the invariants for these attributes. You only need to worry about your new attributes: `_knob` and `_value`.

```
class GSlider(GPath):
    """A class representing a graphical slider.
    MUTABLE ATTRIBUTES:
        _value: the slider value  [float in 0 to 1, inclusive]
    IMMUTABLE ATTRIBUTES:
        _knob:  the slider knob [GEllipse]
    Also, if the slider path is from (x0,y0) to (x1,y1), the _knob is located at
        x = (x1-x0)*_value+x0,     y =(y1-y0)*_value+y0    """
    # DEFINE GETTERS/SETTERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def setValue(self,value):
        """Sets the slider value"""
        assert type(value) == float
        assert 0 <= value and value <= 1
        self._value = value
        self._knob.x = (self.points[2]-self.points[0])*value+self.points[0]
        self._knob.y = (self.points[3]-self.points[1])*value+self.points[1]

    def getValue(self):
        """Gets the slider value"""
        return self._value

    def getKnob(self):
        """Gets the slider knob"""
        return self._knob

    def __init__(self,    x0,    y0,    x1,    y1,    radius = 10):  # Fill in parameters
        """Initializes a slider whose path is from (x0,y0) to (x1,y1)

        The initial _value is 0. The slider line is black, which is the default color
        (so you do not need to set it). However, the knob has a fillcolor of 'blue'.

        Parameter x0: The x-coordinate at the start of the slider  [int]
        Parameter y0: The y-coordinate at the start of the slider  [int]
        Parameter x1: The x-coordinate at the end of the slider  [int]
        Parameter y1: The y-coordinate at the end of the slider  [int]
        Parameter radius: The knob radius [int >= 1]. OPTIONAL (default 10)"""
        super().__init__(points=[x0,y0,x1,y1])
        self._knob = GEllipse(x=x0,y=y0,width=2*radius,height=2*radius,fillcolor='blue')
        self._value = 0



    def draw(self,                    view                  ):     # Fill in parameters
        """Draws this object to the given view.  The knob is ON TOP of the slider path.

        Parameter view: the view to draw to [GView]"""
        super().draw(view)
        self._knob.draw(view)
```

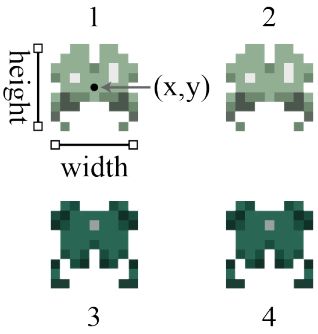
Last Name: _____ First: _____ Netid: _____

4. [12 points] **Iteration**

In Assignment 7, we had you arrange the aliens in a 2-dimensional list. This was to make the assignment easier in the case of determining the rightmost or bottommost alien as they marched across the screen.

However, we did not have to put the aliens in a 2-dimensional list. The location of an alien is determined by the `x` and `y` attributes of the alien, not by its position in the list. Hence we could have put all of the aliens in a 1-dimensional list and asked you to loop over that instead.

Implement the function below. By the "bottom row", we mean the list of aliens that all have the (same) least `y` value. **There is no guarantee on the order of aliens in the original list**. In the wave to the right, the aliens could be in the list in the order 1, 4, 3, 2, where aliens 3 and 4 are the bottom row.



Hint: This function is best done with two separate for-loops. You need to use the attributes `x` and `y` to find the bottom. Once it is located, then you can make the new list of aliens at the bottom. You do not have to worry about the order of aliens in the the list you return.

```
def bottomrow(aliens):
    """Returns a copy of the bottom row of aliens

    Parameter aliens: the 1D list of aliens
    Precondition: aliens is a list of Alien objects (possible empty)"""

    if aliens == []:
        return []

    # Find the minimum y
    min = aliens[0].y
    for alien in aliens:
        if alien.y < min:
            min = alien.y

    result = []
    for alien in aliens:
        if alien.y == min:
            result.append(alien)

    return result
```

Last Name: _____ First: _____ Netid: _____

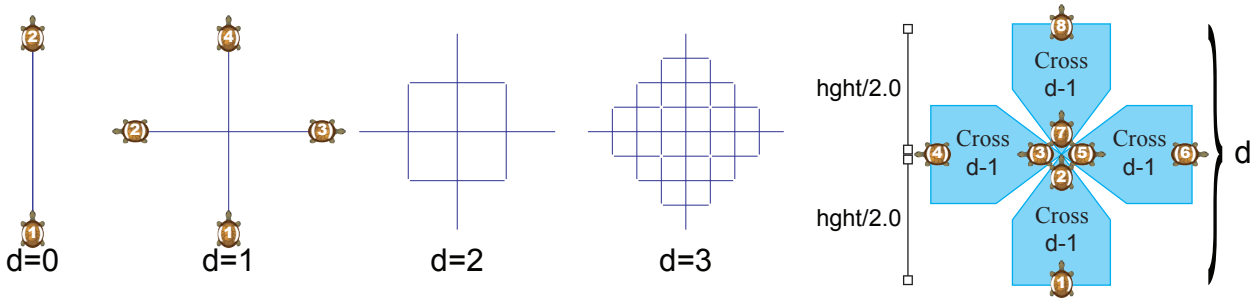
5. [14 points] **Recursion**

Recall the `Turtle` class from Assignment 4. You only need to remember four methods.

Method	Description
`t.forward(n)`	`Moves t forwards n pixels, drawing a line as it goes.`
`t.backward(n)`	`Moves t backwards n pixels, drawing a line as it goes.`
`t.left(a)`	`Turns t left by a degrees.`
`t.right(a)`	`Turns t right by a degrees.`

Implement the function below to draw the recursive shape known as the Greek Cross. At depth 0, this is a vertical line. At higher depths, it is a cross, where each edge of the cross is a Greek Cross of lower depth, as shown below. In all cases, the turtle starts at the bottom of the cross and ends at the top of the cross. **It is okay for the turtle to redraw on top of lines**.

The function below is essentially the helper. You do not need to create a `Window` or `Turtle`.




```
def cross(t, hght, d):
    """Draws Greek Cross of length hght and depth d at turtle's position/heading

    Preconditions: t is a Turtle with drawmode True, hght is a valid length
    (int or float >= 0), and d is a valid depth (int >= 0)"""
    if d == 0:
        t.forward(hght)
        return

    cross(t, hght/2.0, d-1)
    t.left(90)
    cross(t, hght/2.0, d-1)
    t.backward(hght/2.0)
    t.right(180)
    cross(t, hght/2.0, d-1)
    t.backward(hght/2.0)
    t.left(90)
    cross(t, hght/2.0, d-1)
```

Last Name: _____ First: _____ Netid: _____

6. [14 points total] **Testing and Exceptions**

(a) [10 points] Consider the following function specification.

```
def swapcase(s):
    """Returns a copy of s with the case of each letter swapped.

    Characters that are not letters are not effected.

    Precondition: s is a string"""
```

Do not implement this function. Instead, provide a list of at least **five test cases** to test this function. For each test case provide: (1) the function input, (2) the expected output, and (3) an explanation of what makes this test *significantly* different.

There are many possible answers. However, these were the main tests that we had in mind.

Input	Output	Reason
`''`	`''`	The empty list is always important
`'a'`	`'A'`	String with only lower case
`'A'`	`'a'`	String with only upper case
`'Aa'`	`'aA'`	String with a mix of cases
`'Aa!'`	`'aA!'`	String with non letter characters

(b) [4 points] Suppose you are given the following function definitions.

```
def first(n):
    try:
        x = second(n)
    except ArithmeticError:
        x = -n
    return x
```

```
def second(n):
    try:
        y = third(n)
    except ZeroDivisionError:
        y = 100
    return y
```

On the next page, complete the function `third` so that the following are all true.

- `first(0)` returns 100
- `first(n)` **crashes** whenever `n` < 0
- `first(n)` returns `-n` whenever `n` is odd
- `first(n)` returns `n//2` whenever `n` is even.

Moreover, `third` is **not allowed to have any return statement other than the one provided**. You have to produce the functionality above by raising exceptions. You should use the exceptions `Exception`, `ArithmeticError`, and `ZeroDivisionError`. `ArithmeticError` is a subclass of `Exception` and `ZeroDivisionError` is a subclass of `ArithmeticError`.

Hint: Your answer should just consist of if-statements and statements that create errors.

Last Name: _____ First: _____ Netid: _____

```
def third(n):
    """A function that raises a lot of exceptions.

    See the previous page for the output.
    Precondition: n is an int."""


    if n == 0:
        raise ZeroDivisionError()
    elif n < 0:
        raise Exception()
    elif n % 2 == 1:
        raise ArithmeticError()



    # This is the ONLY return allowed in this function
    return n // 2
```

7. [17 points] **Call Frames**

Consider the mutually recursive (they each call each other) functions show below.

```
1  def take(q):
2      if len(q) > 0:
3          return [q[0]]+skip(q[1:])
4      else:
5          return []
```

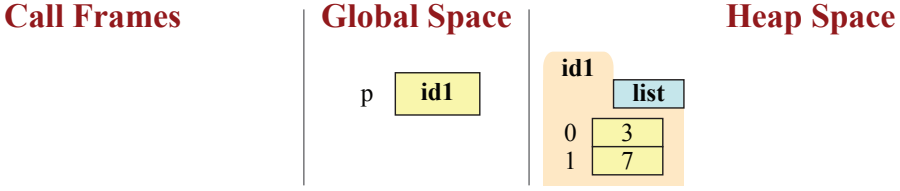
```
6
7  def skip(q):
8      if len(q) < 1:
9          return []
10     return take(q[1:])
```

Suppose that you are given the list `p = [3,7]`. On the next two pages, diagram the execution of the assignment statement

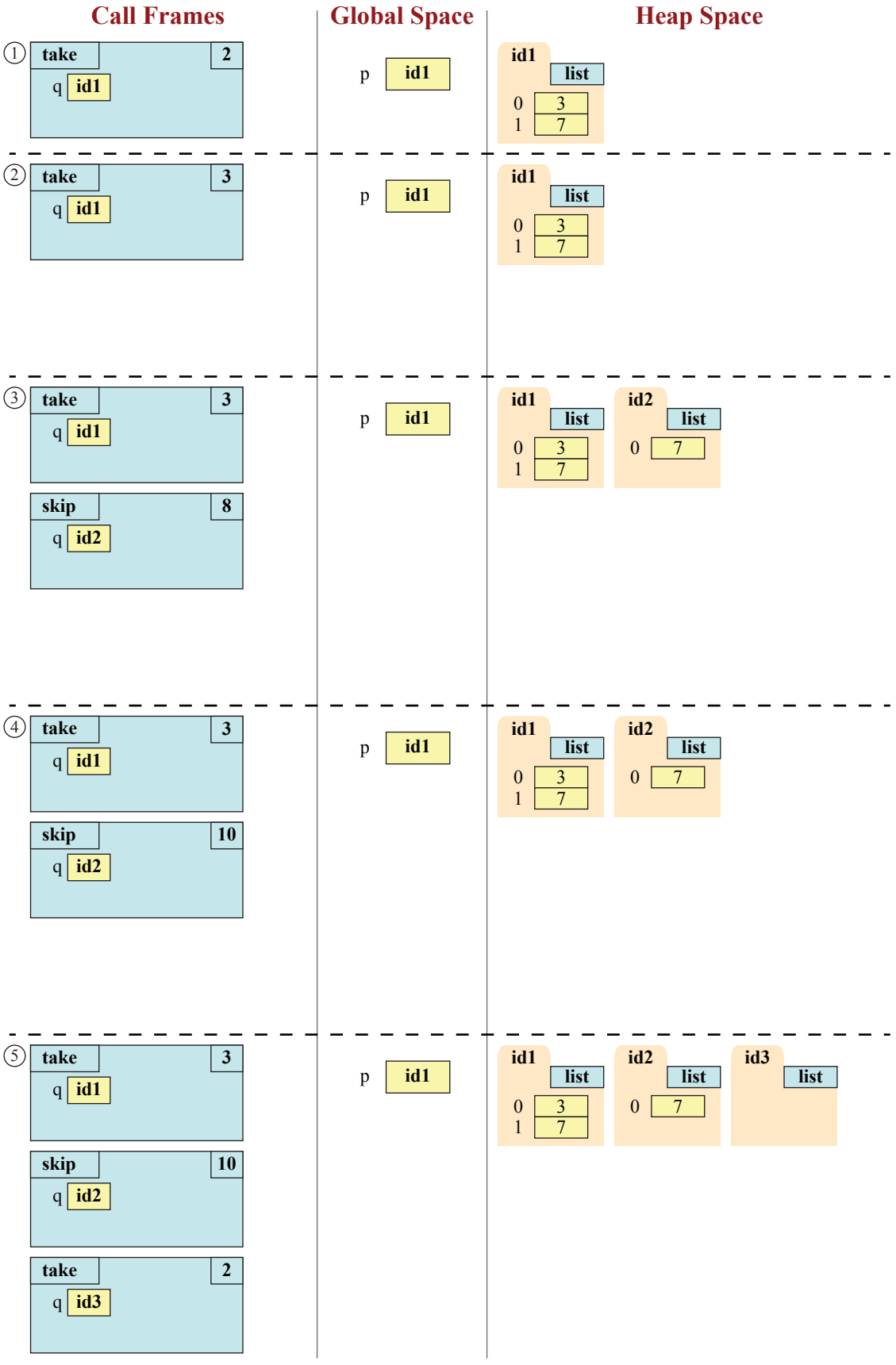
```
>>> x = take(p)
```

We have drawn the variable `p` and its corresponding folder for you below. On the following pages you should draw a new diagram every time a call frame is added or erased, or an instruction counter changes. There are a total of **ten** diagrams to draw. You may write *unchanged* in any of the three spaces if the space does not change at that step.

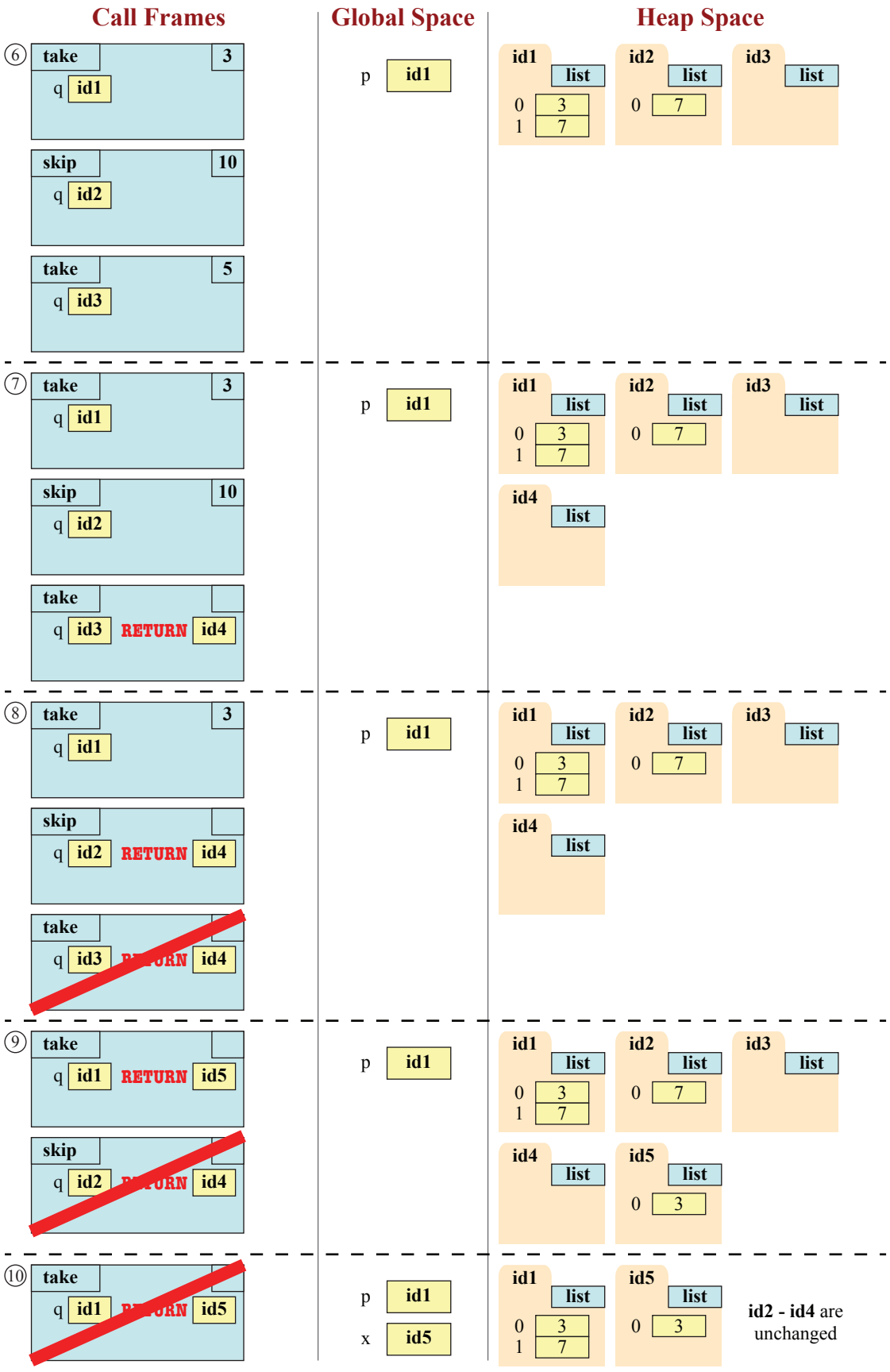
Hint: Remember how slicing works. You are going to create *several* folders in this problem. If you do not want to draw all the folders each time, just tell us which folders are unchanged.



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



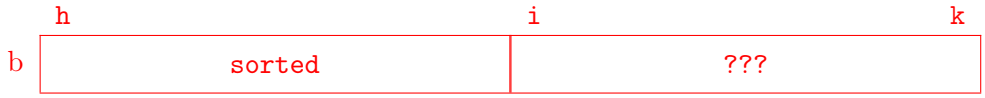
Last Name: _____ First: _____ Netid: _____

8. [14 points total] **Loop Invariants**

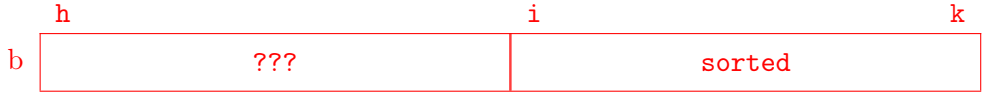
Below are two variations of the insertion sort algorithm from class. The version on the left has been completed for you. Note that this algorithm is a nested loop. The invariant only applies to the outermost loop.

The version on the right has the same precondition and postcondition, but a different loop invariant. It is also missing the code for initialization, the loop condition, and the loop body.

(a) [2 points] Draw the horizontal notation representation for the loop invariant on the left.



(b) [2 points] Draw the horizontal notation representation for the loop invariant on the right.



(c) [10 points] Add the missing code to the function on the right. Like the function on the left, you may use the helper function `swap(b,n,m)` to swap two positions in the list. In class, we extracted the inner loop into its own helper function. You are free to do this if you wish. You do not need to worry about coming up with an invariant for the inner loop. **Solutions that violate the invariant in the outside loop will not receive credit.**

```
def insert1(b,h,k):
    """Sorts the list b[h..k]"""
    # pre: b[h..k] ???
    # Make invariant true at start

    i = h

    # inv: b[h..i-1] sorted, b[i..k] ???

    while i < k+1:

        # Push b[i] into the right position
        # Will not use invariant for loop
        j = i

        while j > h:
            if b[j-1] > b[j]:
                swap(b,j,j-1)
            j = j - 1

        i += 1

    # post: b[h..k] sorted
```

```
def insert2(b,h,k):
    """Sorts the list b[h..k]"""
    # pre: b[h..k] ???
    # Make invariant true at start

    i = k+1

    # inv: b[h..i-1] ???, b[i..k] sorted

    while i > h          :

        # Push b[i] into the right position
        # Will not use invariant for loop
        j = i-1

        while j < k:
            if b[j] > b[j+1]:
                swap(b,j,j+1)
            j = j + 1

        i -= 1

    # post: b[h..k] sorted
```