Last Name: _____ First: _____ Netid: _____ Section _____

CS 1110 Final, December 17th, 2014

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	10	
3	12	
4	18	
5	12	
6	12	
7	18	
8	16	
Total:	100	

The Important First Question:

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

Throughout this exam, there are several questions on sequences (strings, lists, and tuples). All sequences support slicing. In addition, you may find the following 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. [10 points total] **Poutporri**

(a) [2 points] What is the difference between a *statement* and an *expression*?

(b) [3 points] Execute the three statements below. What is printed out? Explain your answer.

```
>>> a = [1,2]
>>> b = a[:]
>>> print (a == b)
>>> print (a is b)
```

(c) [3 points] Below are three expressions. For each one, write its value. If evaluation leads to an error, just say BAD (do not tell us the exception).

`3/2` `True or (5/0 < 1)` `(5/0 < 1) or True`

(d) [2 points] Is the following definition legal? Why or why not? (`abs` is built into Python)

```
def absmax(x,y=0,z):
    """Returns: the maximum absolute value of x, y, and z.
    Precondition: x, y, and z are numbers (int or float)"""
    return max(abs(x),abs(y),abs(z))
```

3. [12 points total] **Testing and Error Handling**

(a) [5 points] Consider the following function from lab.

```
def unique(lst):
    """Returns: The number of unique elements in the list.
    Example: unique([1, 5, 2, 5]) evaluates to 3.
    Precondition: lst is a list on ints (could be empty)."""
```

Do not implement this function. In the space below,provide at least four different test cases to verify that this function is working correctly. For each test case provide: (1) the function input, (2) the expected output, and (3) an explanation of what makes this test *significantly* different.

(b) [7 points] Below are two function definitions using asserts and try-except. Write out the series of print statements displayed for each of the given function calls.

```
def first(n):
    print 'Start first'
    try:
        second(n)
        print 'In first try'
    except:
        print 'In first except'
    print 'Done first'
```

```
def second(n):
    print 'Start second'
    try:
        assert n <= 0, 'is not <= 0'
        print 'In second try'
    except:
        print 'In second except'
    assert n >= 0, 'not >= 0'
    print 'Done second'
```

Function Calls:

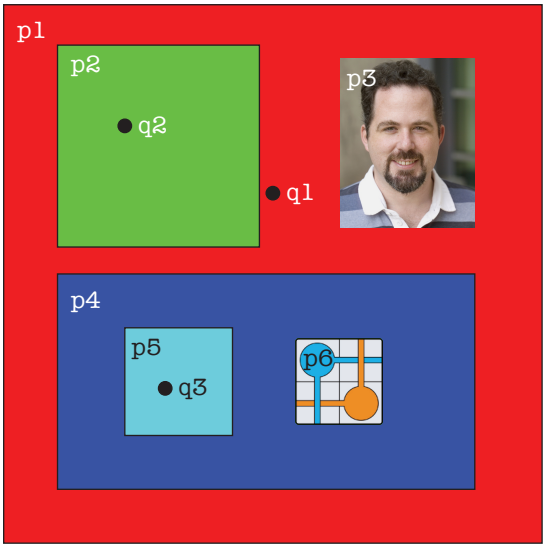
i. `first(-1)`

ii. `first(1)`

4. [18 points] **Classes and Subclasses**

For the next two questions, you will be working with a new class called `GPanel`. The class is a subclass of `GRectangle` from Assignment 7. This class is very similar to an important class in Java that you will see if you continue on to CS 2110.

A `GPanel` is just a `GRectangle` that can contain other instances of `GRectangles` (which can include objects that are `GImage` – itself a subclass of `GRectangle` or even `GPanel`). For example, in the image to the right, the `GPanel p1` contains the `GRectangle p2`, the `GImage p3`, and the `GPanel p4`. Furthermore, the `GPanel p4` contains the `GRectangle p5` and the `GImage p6`. The points `q1`, `q2`, and `q3` are part of the next question, and not important just yet.



On the next page, you will implement the class `GPanel`. We have provided all of the specifications. We have also provided most of the method headers; only `__init__` has an incomplete header. You are to implement all methods **and assert all preconditions**.

In order to implement this class, you will need to remember the attributes and methods of `GRectangle`. They are as follows.

Attribute	Invariant	Description
`x`	`float`	Position of left side.
`y`	`float`	Position of bottom edge.
`width`	`float > 0`	Distance from left side to right side.
`height`	`float > 0`	Distance from bottom edge to top edge.
`linecolor`	instance of `RGB`	Color of rectangle border.
`fillcolor`	instance of `RGB`	Color of rectangle interior.

Method	Description
`contains(x,y)`	**Returns**: True if the point (x,y) is in the rectangle; False otherwise
`draw(view)`	Draws the rectangle to the specified `GView` instance `view`.

There are no hidden attributes or methods that you are aware of. You should also avoid using attributes that you might have remembered from Assignment 7, but are not listed in the tables above (you will not need them).

Finally, recall how the constructor for `GRectangle` works. You provide it with a list of keyword arguments that initialize various attributes. For example, to create a red square anchored at (0,0), use the constructor call

```
GRectangle(x=0,y=0,width=10,height=10,fillcolor=colormodel.RED)
```

The constructor for `GPanel` *does not* work this way. Please read its specification carefully.

```
from game2d import *
import colormodel
class GPanel(GRectangle):
    """Instances are a panel that can store GRectangles (including other GPanels).
    INSTANCE ATTRIBUTES (in addition to those inherited from GRectangle):
        _contents [list of GRectangle (or subclass of GRectangle) objects]."""
    def addContents(self,rect):
        """Adds the GRectangle (or subclass of GRectangle) rect to _contents.
        Precondition: rect is an instance of GRectangle, and is contained inside of
        the GPanel (left is >= panel's left, right is <= panel's right and so on)."""









    def removeContents(self,rect):
        """Removes the GRectangle (or subclass of GRectangle) rect from _contents.
        Precondition: rect is an element of _contents."""



    def clear(self):
        """Removes all elements from _contents"""


    def getContents(self):
        """Returns: a COPY of the list of GRectangles in this GPanel.
        The GRectangles do not need to be copied; only the list."""


    def __init__(                                                ):   # FILL IN
        """Creates a new GPanel of the given dimensions and color.
        Parameters x and y specify the bottom left corner of the panel.  Parameters w
        and h are the width and height.  Parameter color is the fillcolor.
        A new GPanel has no GRectangles stored inside it.
        Precondition: x, y, w, and h are floats with w, h > 0.  color is an instance
        of colormodel.RGB; its default value is colormodel.WHITE."""
```

(**Continued from Previous Page**)

```
def draw(self,view):
    """Draws this GPanel AND ITS CONTENTS to the parameter view.
    The drawing order is important for this method.  First it draws the GPanel
    itself.  Then it draws all of the contents, in the order that they are given
    in the list _contents (so items at the end of the list are on top).
    Precondition: view is an instance of GView."""
```

5. [12 points] **Recursion and Iteration**

We need one more method in `GPanel`. This method takes a point `(x,y)` and returns the top most element containing that point. For example, in the image in question 4, `p1.selected(q1.x,q1.y)` returns `p1`, `p1.selected(q2.x,q2.y)` returns `p2`, and `p1.selected(q3.x,q3.y)` returns `p5`.

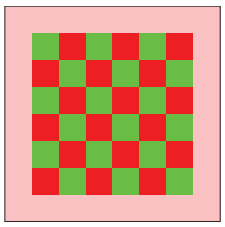
Hint: You will need both recursion and iteration. You *do not* need to assert the precondition.

```
def selected(self,x,y):
    """Returns: the topmost GRectangle containing the point (x,y)
    If (x,y) is not inside this GPanel, it returns None.  Otherwise, it returns
    the top most item (which might be the GPanel itself) containing (x,y).
    Since objects are drawn back to front, top most is the LAST item in _contents
    that contains (x,y).  Furthermore, if that item is itself a GPanel, then you
    must check the contents of that GPanel as well.
    Precondition: x and y are floats."""
```

Last Name: _____ First: _____ Netid: _____ Section _____

6. [12 points] **2-Dimensional Lists**

Recall the class `GRectangle` from Assignment 7 and the previous two problems. In Assignment 7, you had to lay the bricks out on the screen in a pattern. You are going to do something similar here; the function `make_bricks` creates a 2-dimensional list of bricks arranged as shown to the right. In writing this function, keep the following in mind.



- There are `m` columns and rows of squares. In the example above, `m = 8`.
- Each square has side length `SIZE`, and there is no space between them.
- The origin is in the bottom-left corner. x increases to the right; y increases upward.
- Each row in the list contains a single row of bricks, starting at the bottom.
- Squares in columns and rows 0 and m-1 have fillcolor `colormodel.PINK`.
- The bottom, left-most inner square has fillcolor `colormodel.RED`
- Inner squares alternate in a checkerboard of `colormodel.RED` and `colormodel.GREEN`.

```
def make_bricks(m):
    """Returns: a 2-D list of GRectangle objects with the properties above.
    Precondition: m > 0 is an int"""
```

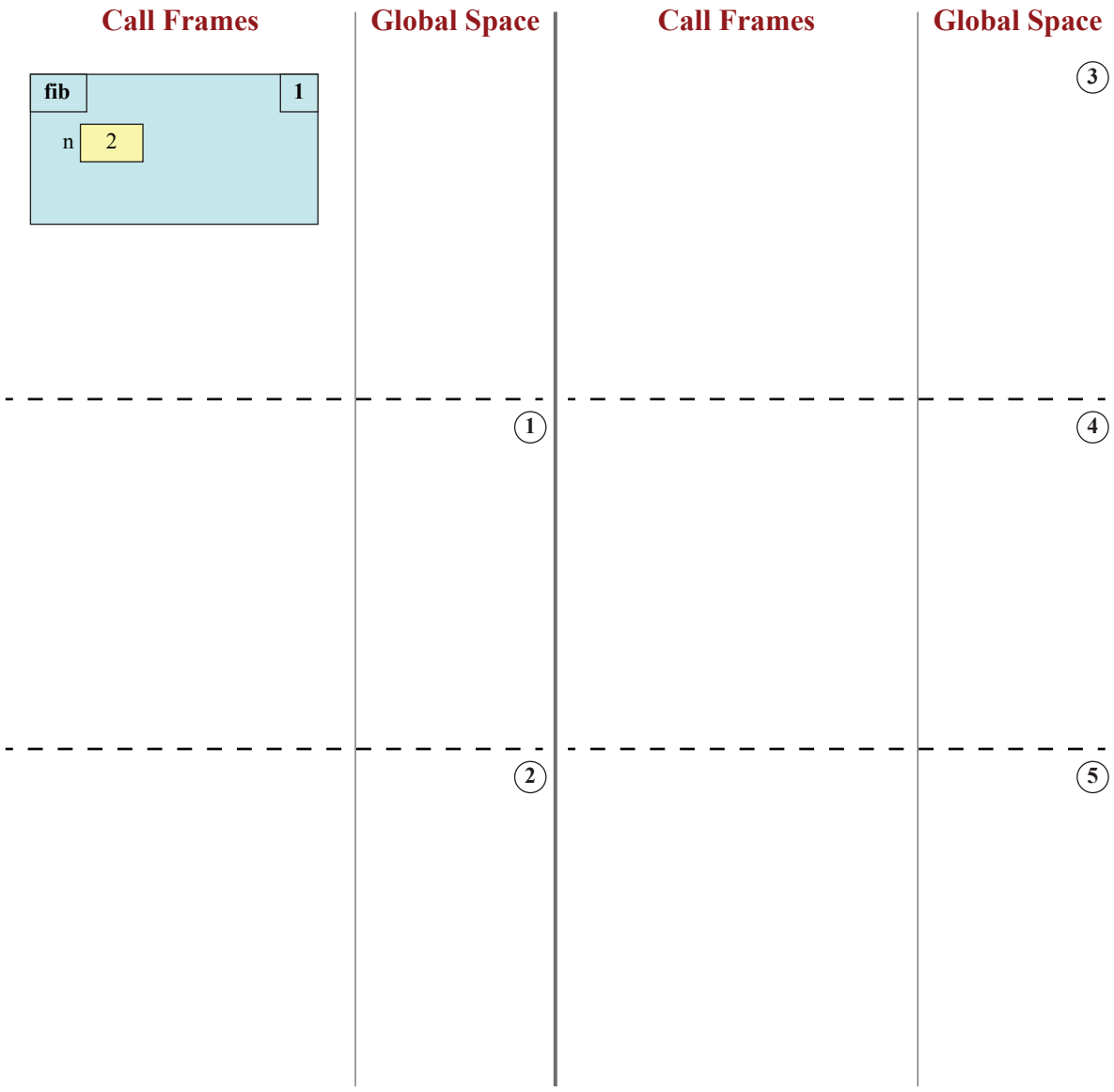
Last Name: _____ First: _____ Netid: _____ Section _____

7. [18 points] **Call Frames**

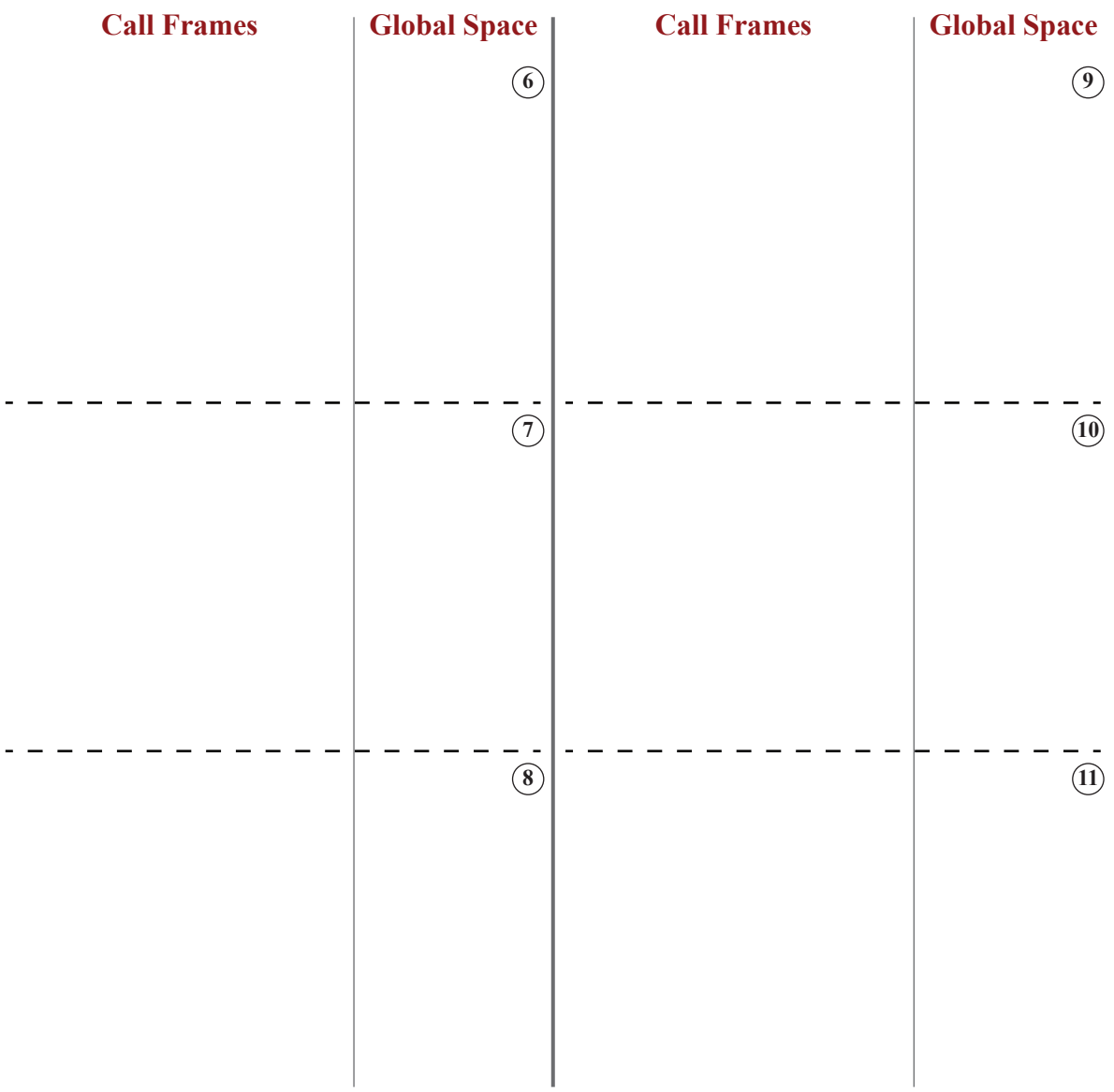
The code to the right is the Fibonacci function from class; line numbers are labelled in red. On this page and the next, diagram the statement `r = fib(2)`. Draw the contents of the call stack and global space, but not heap space. You need a new diagram every time a frame is added or erased, or an instruction counter changes. We have done the first step for you. There are **eleven** more diagrams to draw. Please order your answer according to the numbers given. You may write *unchanged* if a space does not change.

```
def fib(n)
    if n == 0 or n == 1:    1
        return 1            2

    one = fib(n-1)          3
    two = fib(n-2)          4
    return one+two          5
```



Last Name: _____ First: _____ Netid: _____ Section _____



8. [16 points total] **Loop Invariants**

   On the next page are two variations of the Dutch National Flag algorithm from class. The one on the left has been completed for you. The second algorithm is identical to the first except that it has a different loop invariant. It is currently missing the code for initialization, the loop condition, and the body of the loop (all other code is provided).

   (a) [3 points] Draw the horizontal notation representation for the loop invariant on the left.

(b) [3 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. It is not enough for your code to be correct. **To get credit**, you must satisfy the loop invariant.

```
def dnf(b,h,k):
    """Returns: Divisions i, j of dnf
    Pre: h, k are positions in list b."""
    # Make invariant true at start

    t = h

    j = k+1

    i = k+1

    # inv: b[h..t-1] < 0, b[t..i-1] ???,
    # b[i..j-1] = 0, and b[j..k] > 0


    while t < i:
        if b[i-1] < 0:
            swap(b,i-1,t)

            t= t+1

        elif b[i-1] == 0:
            i= i-1

        else:
            swap(b,i-1,j-1)

            i= i-1

            j= j-1

    # post: b[h..i-1] < 0, b[i..j-1] = 0,
    # and b[j..k] > 0
    return (i,j)
```

```
def dnf(b,h,k):
    """Returns: Divisions i, j of dnf
    Pre: h, k are positions in list b."""
    # Make invariant true at start






    # inv: b[h..i-1] < 0, b[i..j-1] = 0,
    # b[j..t-1] ???, and b[t..k] > 0


    while _______________:








    # post: b[h..i-1] < 0, b[i..j-1] = 0,
    # and b[j..k] > 0
    return (i,j)
```