Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 2 November 19th, 2020

This 90-minute exam has 5 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
```

You should not use while-loops on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, try-except, lists, for-loops, recursion and so on).

Question	Points	Score
1	2	
2	20	
3	23	
4	25	
5	30	
Total:	100	

The Important First Question:

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

Last Name: _____ First: _____ Netid: _____

Reference Sheet

String Operations

Operation	Description
`len(s)`	**Returns**: Number of characters in `s`; it can be 0.
`a in s`	**Returns**: True if the substring `a` is in `s`; False otherwise.
`a*n`	**Returns**: The concatenation of `n` copies of `a`: `a`+`a`+$\cdots$+`a`.
`s.find(s1)`	**Returns**: Index of FIRST occurrence of `s1` in `s` (-1 if `s1` is not in s).
`s.count(s1)`	**Returns**: Number of (non-overlapping) occurrences of `s1` in `s`.
`s.islower()`	**Returns**: True if `s` is *has at least one letter* and all letters are lower case; it returns False otherwise (e.g. `'a123'` is True but `'123'` is False).
`s.isupper()`	**Returns**: True if `s` is *has at least one letter* and all letters are uppper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).
`s.isalpha()`	**Returns**: True if `s` is *not empty* and its elements are all letters; it returns False otherwise.
`s.isdigit()`	**Returns**: True if `s` is *not empty* and its elements are all numbers; it returns False otherwise.
`s.isalnum()`	**Returns**: True if `s` is *not empty* and its elements are all letters or numbers; it returns False otherwise.

List Operations

Operation	Description
`len(x)`	**Returns**: Number of elements in list `x`; it can be 0.
`y in x`	**Returns**: True if `y` is in list `x`; False otherwise.
`x.index(y)`	**Returns**: Index of FIRST occurrence of `y` in `x` (error if `y` is not in `x`).
`x.count(y)`	**Returns**: the number of times `y` appears in list `x`.
`x.append(y)`	Adds `y` to the end of list `x`.
`x.insert(i,y)`	Inserts `y` at position `i` in `x`. Elements after `i` are shifted to the right.
`x.remove(y)`	Removes first item from the list equal to `y`. (error if `y` is not in `x`).

Dictionary Operations

Function or Method	Description
`len(d)`	**Returns**: number of keys in dictionary `d`; it can be 0.
`y in d`	**Returns**: True if `y` is a key `d`; False otherwise.
`d[k] = v`	Assigns value `v` to the key `k` in `d`.
`del d[k]`	Deletes the key `k` (and its value) from the dictionary `d`.
`d.clear()`	Removes all keys (and values) from the dictionary `d`.

Last Name: _____ First: _____ Netid: _____

2. [20 points total] **Iteration**.

   Implement the functions below according to their specification using for-loops. You **do not** need to enforce preconditions.

   (a) [7 points]

```
def sumfold(lst):
    """MODIFIES the list to contain the accumulated sum.
    Each element at position i becomes the sum up to and including that position.
    Example: If a = [0,1,2,3,4], then sumfold(a) changes a to [0,1,3,6,10]
    Precondition: lst is a nonempty list of integers"""

    sum = 0   # Intermedite accumulator

    # Mutability requires loops over positions
    for pos in range(len(lst)):
        sum = sum + lst[pos]
        lst[pos] = sum
    # No return value
```

   (b) [13 points]

```
def zerocols(table):
    """Returns a list of (positions of) all zero columns in the given table.
    Columns identified must be all zero; the presence of one zero is not enough.
    Example: zerocols([[1,0,2],[0,0,0],[3,0,4]]) returns [1]
             zerocols([[0,0],[0,0]]) returns [0,1]
             zerocols([[1,0],[0,2]]) returns []
    Precondition: table is a nonempty rectangular 2d list of numbers"""

    result = []

    # Loop over all columns
    for pos in range(len(table[0])):
        # Search for non-zero value
        good = True
        for x in table:
            if x[pos] != 0:
                good = False

        # Add to accum if all 0
        if good:
            result.append(pos)

    return result
```

Last Name: _____ First: _____ Netid: _____

3. [23 points total] **Recursion**.

   Use recursion to implement the following functions. Solutions using for-loops will receive no credit. You **do not** need to enforce the preconditions.

   (a) [10 points]

```
def prefix(s):
    """Returns the prefix (identical characters at the start) length of s
    Example: prefix('abc') returns 1 as the prefix is 'a'
             prefix('xxxxxxyzx') returns 6 as the prefix is 'xxxxxx'
             prefix('') returns 0 as the string is empty
    Precondition: s is a (possibly empty) string of lowercase letters"""

    if s == '':
        return 0
    elif len(s) == 1:
        return 1

    left  = prefix(s[:1])
    right = prefix(s[1:])

    if s[0] == s[1]:
        return left+rght

    return left
```

   (b) [13 points] An *inverted string* is a dictionary whose keys are characters and whose values are lists of positions. For example, the string `'hello'` is inverted as the dictionary

```
{ 'h':[0], 'e':[1], 'l':[2,3], 'o':[4] }
```

   Hint: How you divide matters on this problem. Do *not* pull off one element at the start.

```
def invert(s):
    """Returns an inverted string representing s
    Example: invert('abcac') returns {'a':[0,3], 'b':[1], 'c':[2,4]}.
             invert('') is {}.
    Precondition: s is a (possibly empty) string"""
    if s == '':
        return {}

    left  = invert(s[:-1])

    # Remove right to make combination easier
    char = s[-1]
    pos  = len(s)-1

    if char in left:
        left[char].append(pos)
    else:
        left[char] = [pos]

    return left
```

Last Name: _____ First: _____ Netid: _____

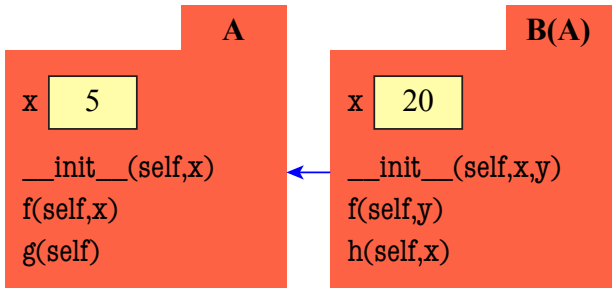
4. [25 points total] **Folders and Name Resolution**

Consider the two (undocumented) classes below, together with their line numbers.

```
1  class A(object):
2      x = 5
3
4      def __init__(self,x):
5          self.y = x
6          self.f(x)
7
8      def f(self,x):
9          self.x = x+x
10
11     def g(self):
12         return 2*self.y
```

```
13 class B(A):
14     x = 20
15
16     def __init__(self,x,y):
17         super().__init__(y)
18         self.y = x
19
20     def f(self,y):
21         self.x = self.x-y
22
23     def h(self,x):
24         return x+self.y//2
```

(a) [5 points] Draw the class folders in the heap for these two classes.



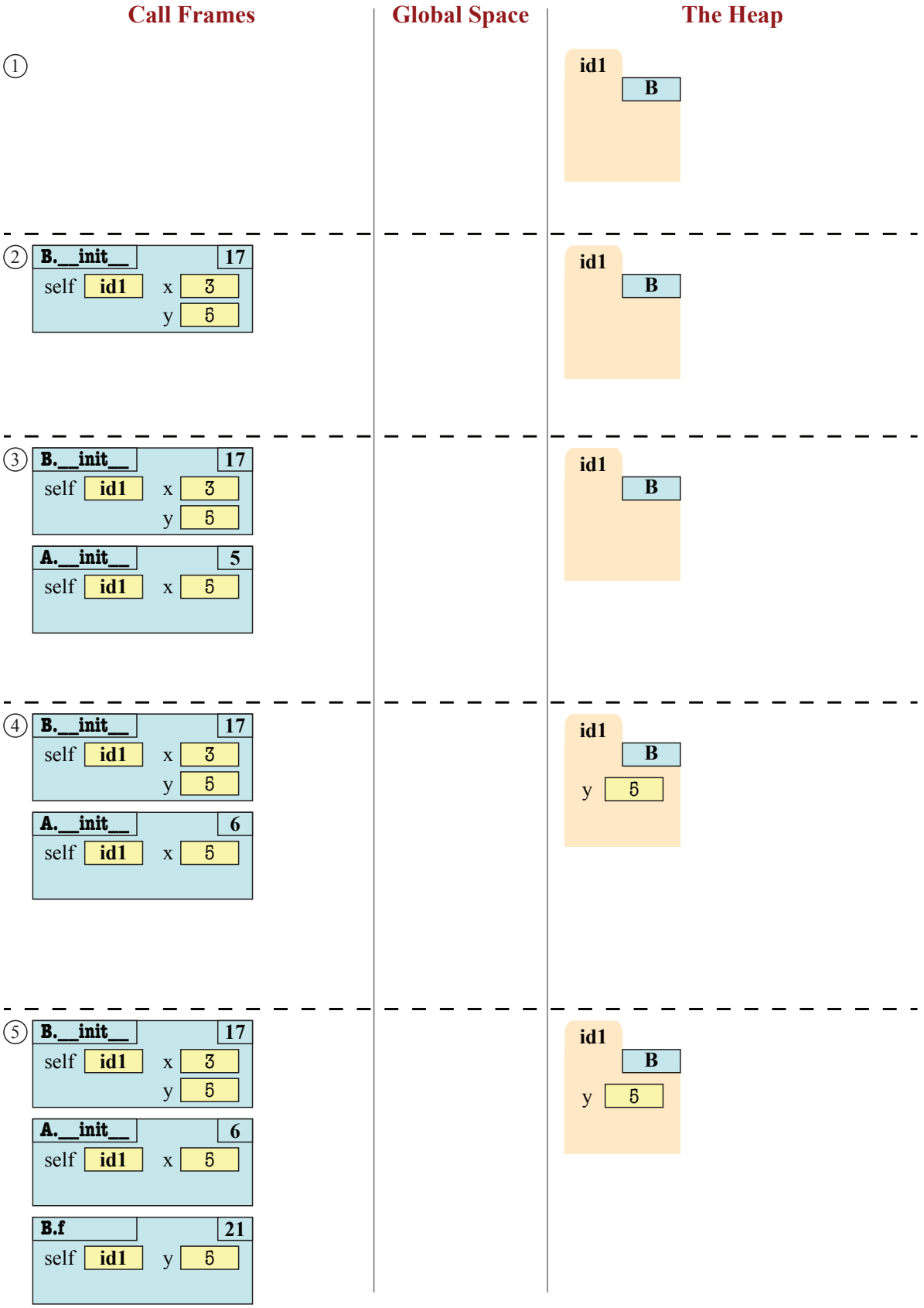
(b) [20 points] On the next two pages, diagram the call

```
>>> b = B(3,5)
```

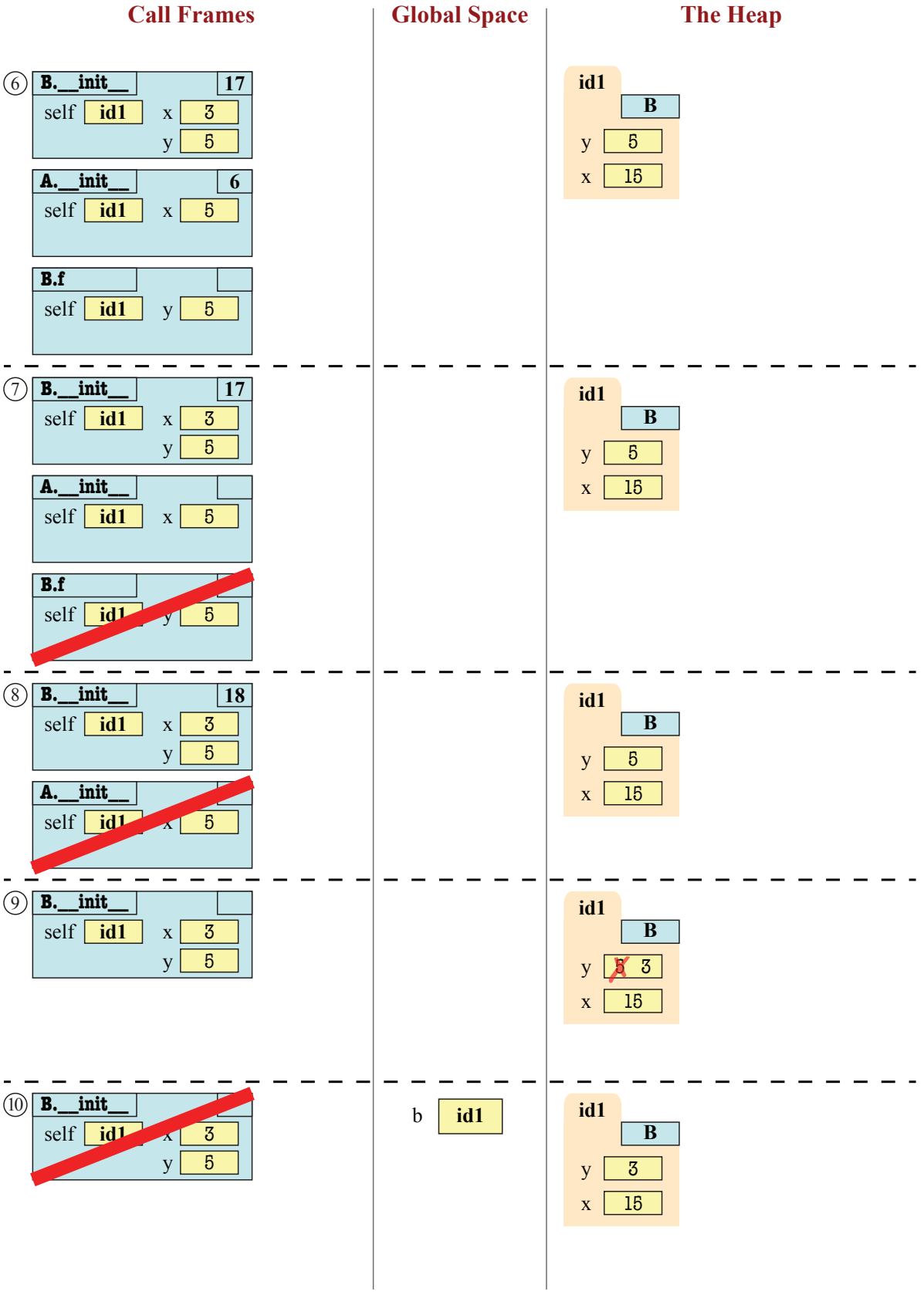
You will need **ten diagrams**. Draw the call stack, global space and the heap. If the contents of any space are unchanged between diagrams, you may write *unchanged*. You do not need to draw the class folders from part (a).

When diagramming a constructor, you should follow the rules from Assignment 5. Remember that `__init__` is a helper to a constructor but it is not the same as the constructor. In particular, there is an important first step before you create the call frame.

Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



5. [30 points total] **Classes and Subclasses**

In this problem, you will create a class `Date` (which represents a year, month, day) and its subclass `DateTime` which includes an hour and minute of the day.

While most of the attributes are integers, it will store the month as a 3-letter abbreviation (e.g. `'Jan'`, `'Feb'`, `'Mar'`, `'Apr'`, `'May'`, `'Jun'`, `'Jul'`, `'Aug'`, `'Sep'`, `'Oct'`, `'Nov'`, or `'Dec'`). Remember that `'Apr'`, `'Jun'`, `'Sep'`, and `'Nov'` have 30 days, `'Feb'` has either 28 or 29, and all others have 31 days. The attributes of the two classes are as follows:

`Date`

Attribute	**Invariant**	**Category**
`MONTHS`	list of 3-letter month abbreviations in order	CLASS attribute
`DAYS`	dictionary from months to number of days	CLASS attribute
`_year`	int >= 2000	**Immutable** instance attribute
`_month`	3-letter string abbreviation	**Immutable** instance attribute
`_day`	int that is a valid day of `_month`	**Mutable** instance attribute

`DateTime` (in addition to those inherited)

Attribute	**Invariant**	**Category**
`_hour`	int in range 0..23	**Mutable** instance attribute
`_minute`	int in range 0..59	**Mutable** instance attribute

Instructions: On the next four pages, you are to do the following:

1. Fill in the missing information in each class header.
2. Add any necessary class attributes
3. Add getters and setters as appropriate for the instance attributes
4. Fill in the parameters of each method (beyond the getters and setters)
5. Implement each method according to the specification.
6. Enforce any preconditions in these methods using asserts

We have not added headers for the getters and setters. You are to write these from scratch. However, **you are not expected to write specifications for them**. For the other methods, pay attention to the provided specifications. The only parameters are those in the preconditions.

The class `DateTime` **may not** use any attribute or getter/setter inherited from `Date`. It may only use `super()` to access overridden methods.

You should enforce preconditions with `assert` unless you are given a specific error to use instead. Type-based preconditions should all be managed with `isinstance` and not `type`.

Finally, there is the matter of February. In the `DAYS` class attribute, you should consider February as having 28 days, and ignore leap years. However, you should *not* ignore leap years (February has 29 days) when enforcing the invariant of the `_day` attribute. To help you with that you are free to use the following helper function (*you do not need to implement this*).

```
def isleapyear(y):
    """Returns True if y is a leap year. False otherwise
       Precondition: y is an int >= 0"""
```

Last Name: _____ First: _____ Netid: _____

(a) [18 points] The class `Date`

```
class Date(object):                                          # Fill in missing part
    """A class representing a month, day and year

    Attribute MONTHS: A CLASS ATTRIBUTE list of all month abbreviations in order
    Attribute DAYS:   A CLASS ATTRIBUTE that is a dictionary. Keys are the strings
                      from MONTHS; values are days in that month ('Feb' is 28 days)"""
    # Attribute _year: The represented year. An int >= 2000 (IMMUTABLE)
    # Attribute _month: The month. A valid 3-letter string from MONTHS (IMMUTABLE)
    # Attribute _day: The day. An int representing a valid day of _month (MUTABLE)

    # CLASS ATTRIBUTES.
    MONTHS = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

    DAYS = {'Jan':31,'Feb':28,'Mar':31,'Apr':30,'May':31,'Jun':30,'Jul':31,'Aug':31,
            'Sep':30,'Oct':31,'Nov':30,'Dec':31 }




    # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def getYear(self):
        """Returns the year of this date"""
        return self._year

    def getMonth(self):
        """Returns the month of this date"""
        return self._month

    def getDay(self):
        """Returns the day of this date"""
        return self._day

    def setDay(self,value):
        """Sets the day of this date

        Parameter value: The new day
        Precondition: value is a valid day in the month
        assert isinstance(value,int)
        if self._month == 'Feb' and isleapyear(self._year):
            assert value > 0 and value <= 29
        else:
            assert value > 0 and value <= self.DAYS[self._month]
        self._days = value
```

Last Name: _____ First: _____ Netid: _____

```
# Class Date (CONTINUED).

def __init__(self,    y,    m,    d):                          # Fill in missing part
    """Initializes a new date for the given month, day, and year

    Precondition: y is an int >= 2000 for the year
    Precondition: m is a 3-letter string for a valid month
    Precondition: d is an int and a valid day for month m"""

    assert assert isinstance(y,int)
    assert y >= 2000
    assert m in self.MONTHS
    self._year  = y
    self._month = m
    self.setDay(d)  # Enforces the precondition




def __str__(self):                                             # Fill in missing part
    """Returns a string representation of this date.

    The representation is month day, year like this: 'Jan 2, 2002' """

    return self._month+' '+str(self._day)+', '+str(self.year)




def __lt__(self,other):                                        # Fill in missing part
    """Returns True if this date happened before other (False otherwise)

    Precondition: other is a Date
    This method causes a TypeError if the precondition is violated."""
    # IMPORTANT: You are limited to 20 lines.  Do NOT brute force this.

    if not isinstance(other,Date):
        raise TypeError()

    m1 = Date.MONTHS.index(self.getMonth())     # self month as number
    m2 = Date.MONTHS.index(other.getMonth())    # other month as number

    if self.getYear() != other.getYear():
        return self.getYear() < other.getYear()
    elif m1 != m2:
        return m1 < m2
    return self.getDay() < other.getDay()
```

(b) [12 points] The class `DateTime`.

```
class DateTime(Date):                                    # Fill in missing part
    """A class representing a month, day and year, plus time of day (hours, minutes)"""
    # Attribute _hour: The hour of the day. An int in range 0..23 (MUTABLE)
    # Attribute _minute: The minute of the hour. An int in range 0..59 (MUTABLE)

    # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def getHour(self):
        """Returns the hour of the day"""
        return self._hour

    def setHour(self,value):
        """Sets the hour of the day

        Parameter value: The new hour
        Precondition: hour is an int in 0..23
        assert isinstance(value,int)
        assert value >= 0 and value < 24
        self._hour = value

    def getMinute(self):
        """Returns the minute of the hour"""
        return self._minute

    def setMinute(self,value):
        """Sets the hour of the day

        Parameter value: The new hour
        Precondition: hour is an int in 0..23
        assert isinstance(value,int)
        assert value >= 0 and value < 60
        self._minute = value
```

Last Name: _____ First: _____ Netid: _____

```
# Class Date (CONTINUED).
# REMEMBER: This page may not use any attributes or getters/setters from Date.


def __init__(self,  y,  m,  d,  hr=0,  mn=0):          # Fill in missing part
    """Initializes a new datetime for the given month, day, year, hour and minute

    This method adds two additional (default) parameters to the initialize for
    Date.  They are hr (for hour) and mn (for minute).

    Precondition: y is an int >= 2000 for the year
    Precondition: m is a 3-letter string for a valid month
    Precondition: d is an int and a valid day for month m
    Precondition: hr is an int in the range 0..23 (OPTIONAL; default 0)
    Precondition: mn is an int in the range 0..59 (OPTIONAL; default 0)"""

    super().__init__(y,m,d)
    self.setHour(hr)
    self.setMinute(mn)









def __str__(self):                                      # Fill in missing part
    """Returns a string representation of this DateTime object

    The representation is 'hh:mm on month day, year' like this: '9:45 on Jan 2, 2002'
    Single digit minutes should be padded with 0s.  Hours do not need to be padded."""

    if self._minute < 10:
        time = str(self._hour)+':0'+str(self._minute)
    else:
        time = str(self._hour)+':'+str(self._minute)
    return time+' on '+super().__str__()
```