CS 111: Introduction to Computer Science

Winter 2024 Semester

Professors: Brayden Connole, Steve Richardson, Daniel Richins, Tom Stephens

MIDTERM 2
KEY

You may not write on the test booklet as you work through your problems. Please do all your work on scratch paper.

(3 points) **Recursive Objects: Tree**

A *Binary Search Tree* is a sorted tree data structure where each Tree node only has two branches, which are called *left* and *right*. Each Tree node's **left** branch contains zero or more Tree nodes with labels **smaller** than its label. Each Tree node's **right** branch contains zero or more Tree nodes with labels **larger** than its label.
Here is an example Binary Search Tree:
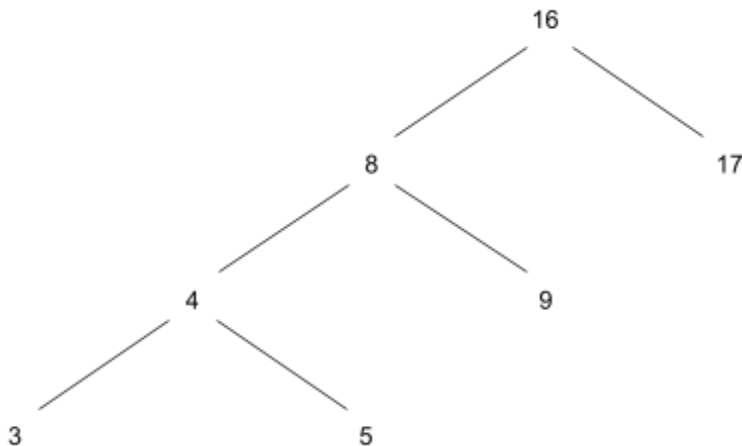
The code below defines a class called `SortedTree` that is a recursive object and has a `left` and `right` branch.

```python
class SortedTree:
    """A binary tree."""
    empty = ()

    def __init__(self, label, left=empty, right=empty):
        self.label = label
        self.left = left
        self.right = right

    def is_leaf(self):
        """
        >>> t = SortedTree(1)
        >>> t.is_leaf()
        True
        >>> t = SortedTree(5, SortedTree(3), SortedTree(7))
        >>> t.is_leaf()
        False
        """
        return not self.left and not self.right
```

We want to create a function for a `SortedTree` sorted as a Binary Search Tree to return a list of the labels of the tree in sorted order – smallest to greatest. For example, the example tree (above) should return the list [3, 4, 5, 8, 9, 16, 17].

Here is the code with missing parts.

```python
def inorder(sorted_tree):
    """ Argument sorted_tree must be a sorted binary tree
    >>> t = SortedTree(8, SortedTree(7), SortedTree(9))
    >>> inorder(t)
    [7, 8, 9]
    """
    nodes = []
    if sorted_tree != SortedTree.empty:
        nodes += inorder(____(a)____)
        nodes.append(____(b)____)
        nodes += inorder(____(c)____)
    return nodes
```

Note that the doc string above is for this tree:
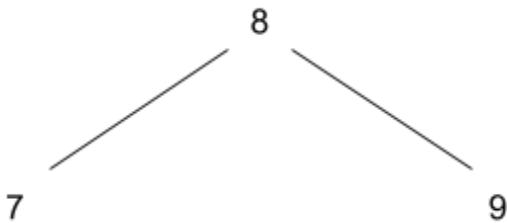
(1) (1 pt) What line of code could go in blank (a)?
  A. sorted_tree.label
  B. sorted_tree.left
  C. sorted_tree.right

(2) (1 pt) What line of code could go in blank (b)?
  A. sorted_tree.label
  B. sorted_tree.left
  C. sorted_tree.right

(3) (1 pt) What line of code could go in blank (c)?
  A. sorted_tree.label
  B. sorted_tree.left
  C. sorted_tree.right

(7 points) **Regular Expressions**

(4) (2 pts) Given the pattern "[def]+QY?[4-6]", which of the following would be matched? **Select the *two* that are correct**.

    A. "def456"

    B. "dQ5"

    C. "dey4"

    D. "defQY6"

    E. "xyzdQY4"

    F. "efY6"

    G. "DQY4"

(5) (3 pts) Which of the following patterns would match "def" and "ddeff" but not "ef"? **Select the *three* that are correct**.

    A. "d+ef+"

    B. "[def]+"

    C. "d?[ef]"

    D. "d[def]+"

    E. ".*ef.*"

    F. ".?def.?"

(6) (2 pts) Which of the following patterns would match "Matthew", "Mark", and "John" but not "Luke"? **Select the *two* that are correct**.

    A. "(matthew|mark|john)?"

    B. "([A-K]|[M-Z])\w*"

    C. "[MJ]?\w+"

    D. "((Ma|Jo)[trh].*)?"

    E. "([MJ][ao][trh][tkn]\d+.*)?"

    F. "\w[aeiou]\w+"

(6 points) Searching a Sorted List

Given a sorted list of `n` integers, how do we determine if some target value is in the list? We could iterate and search through the entire list until we find the target or exhaust the search. However, this can be very slow if we are working with a large list.

A binary search is performed by using a Divide and Conquer algorithm. Divide and Conquer algorithms divide the list into two smaller lists and then recursively operate on the sublists. With binary search, the algorithm divides the list into two sublists, where one sublist has smaller numbers and the second sublist has larger numbers. Binary search then discards the sublist that is too big or too small for the target value being searched. Binary search only continues to search the sublist that might actually contain the target value. This decision to discard a sublist is made with just one comparison.

For example, given the following sorted list: [1, 2, 3, 4, 5, 6, 7, 8], there are eight elements in sorted order. If we are trying to search this list for the value 6, then Binary Search will first split the list into two sublists: [1, 2, 3, 4] and [5, 6, 7, 8]. The target value, 6, can only be found in the numbers >= 5, so we discard the first sublist and recursively search on the second sublist.

The following code is a *recursive* implementation of Binary Search (with some lines missing). Determine what should fill in the blanks.

```
# Recursive implementation of the binary search algorithm to return
# the position of `target_value` in sub-list nums[low_index...high_index]
def binarySearch(sorted_list, low_index, high_index, target_value):

    # Base condition (search space is exhausted)
    if low_index > high_index:
        return -1

    # find the mid-value in the search space and compares it with the target
    mid_index = (high_index + low_index) // 2

    # Base condition (target value is found)
    if target_value == sorted_list[mid_index]:
        return __(a)__

 # discard all elements in the right search space, including the middle element
    elif target_value < sorted_list[mid_index]:
        return binarySearch(____(b)____)

 # discard all elements in the left search space, including the middle element
    else:
        return binarySearch(____(c)____)
```

(7) (1 pt) What line of code could go in blank (a)?

    A. `sorted_list`
    B. `high_index`
    C. `low_index`
    D. `mid_index`
    E. `sorted_list[low_index]`
    F. `sorted_list[mid_index]`
    G. `sorted_list[high_index]`

(8) (1 pt) What line of code could go in blank (b)?

    A. `sorted_list, mid_index + 1, high_index, target_value`
    B. `sorted_list, low_index, high_index, target_value`
    C. `sorted_list, low_index, mid_index - 1, target_value`
    D. `sorted_list, high_index, low_index, target_value`

(9) (1 pt) What line of code could go in blank (c)?

    A. `sorted_list, mid_index + 1, high_index, target_value`
    B. `sorted_list, low_index, high_index, target_value`
    C. `sorted_list, low_index, mid_index - 1, target_value`
    D. `sorted_list, high_index, low_index, target_value`

Recall from our class discussion that all recursive algorithms can also be written iteratively (and vice versa). The code on this page is an *iterative* implementation of Binary Search (with some lines missing).

```python
# Function to determine if a `target_value` exists in the sorted list
# or not using a binary search algorithm
def lookup(sorted_list, target_value):
    # search space is nums[left…right]
    (low_index, high_index) = (0, len(sorted_list) - 1)
    # loop till the search space is exhausted
    while low_index <= high_index:
     # find the mid-value in the list and compares it with the target value
        mid_index = (low_index + high_index) // 2
        # target value is found
        if target_value == sorted_list[mid_index]:
            return __(d)
    # discard elements in the right search space, including the middle element
        elif target_value < sorted_list[mid_index]:
            high_index = __(e)
    # discard elements in the left search space, including the middle element
        else:
            low_index = __(f)
     # `target_value` doesn't exist in the list
    return -1
```

(10) (1 pt) What line of code could go in blank (d)?
      A. `mid_index / 2`
      B. `mid_index`
      C. `mid_index * 2`

(11) (1 pt) What line of code could go in blank (e)?
      A. `mid_index`
      B. `mid_index + 1`
      C. `mid_index - 1`
      D. `mid_index * 2`

(12) (1 pt) What line of code could go in blank (f)?
      A. `mid_index`
      B. `mid_index + 1`
      C. `mid_index - 1`
      D. `mid_index * 2`

(5 points) Generators for Recursive Objects: Link

The code below defines a class called `Link` that is a recursive object. Note that the member variable that points to the next `Link` is called `next`, instead of `rest` as was shown in the lecture slides (using the name "next" is common practice with linked lists). The code uses `label` instead of `first` as shown in the slides.

```python
class Link:
    """A linked list."""
    empty = ()

    def __init__(self, label, next=empty):
        assert next is Link.empty or isinstance(next, Link)
        self.label = label
        self.next = next
```

The function below takes an argument of type `Link` and creates a generator that will give the `label` of each `Link` in the order of the linked list.

```python
def inorder(linked_list):
    """
    >>> l_lst = Link(4, Link(8, Link(15, Link(16, Link(23, Link(42))))))
    >>> order_generator = inorder(l_lst)
    >>> next(order_generator)  # First call
    4
    >>> next(order_generator)  # Second call
    8
    >>> next(order_generator)  # Third call
    15
    """
    lst = linked_list
    while __(a)__:
        yield __(b)__
        ____(c)__
```

(13) (1 pt) What line of code could go in blank (a)?

    A. `lst is Link.empty`

    B. `lst is not Link.empty`

    C. `lst.next is not Link.empty`

    D. `lst.next is Link.empty`

(14) (2 pts) What line of code could go in blank (b)?

    A. `linked_list`

    B. `lst.label`

    C. `lst.next`

    D. `lst`

(15) (2 pts) What line of code could go in blank (c)?

    A. `lst = linked_list.next`

    B. `lst = linked_list.value`

    C. `lst = lst.next`

    D. `lst = lst.value`

    E. `lst = linked_list`

## (7 points) WWPD Recursive Objects: Doubly Linked List

A *doubly* linked list is a list where each List node has a reference to both the next List node and the previous List node, which are stored as `next` and `prev` respectively in the object. See the code below and answer the questions.

```
class DoubleLink:
    """A doubly linked list."""
    empty = ()

    def __init__(self, label, next=empty, prev=empty):
        assert next is DoubleLink.empty or isinstance(next, DoubleLink)
        assert prev is DoubleLink.empty or isinstance(prev, DoubleLink)
        self.label = label
        self.next = next
        if next is not DoubleLink.empty:
            next.prev = self
        self.prev = prev
        if prev is not DoubleLink.empty:
            prev.next = self
```

Suppose we run the following code to create a `DoubleLink` list (graphically shown below)

```
>>> first_course = DoubleLink("CS110")
>>> last_course = DoubleLink("CS235")
>>> second_course = DoubleLink("CS111", last_course, first_course)
>>> start = first_course
```
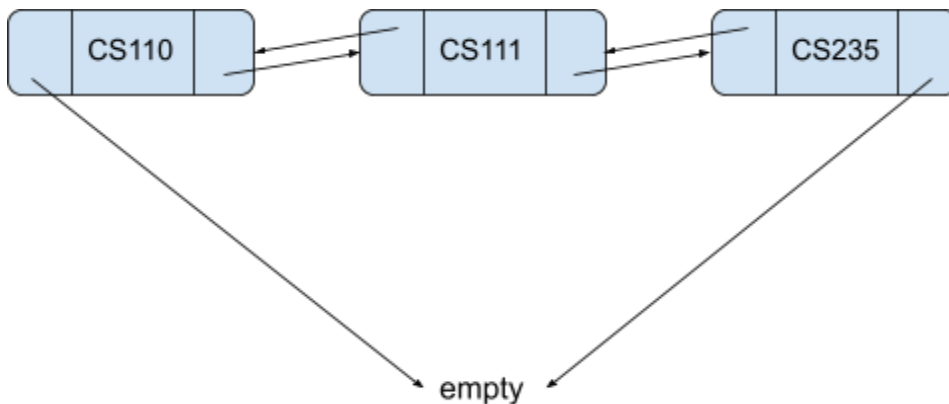
(16) (1 pt) What would Python display? >>> `start.label`
- A. CS235
- B. CS111
- C. **CS110**

(17) (1 pt) What would Python display? >>> `start.next.label`
- A. CS235
- B. **CS111**
- C. CS110

(18) (1 pt) What would Python display? >>> `start.next.next.label`
- A. **CS235**
- B. CS111
- C. CS110

(19) (2 pts) What would Python display? >>> `start.next.next.prev.label`
- A. CS235
- B. **CS111**
- C. CS110

(20) (2 pts) What would Python display? >>> `start.next.next.prev.prev.label`
- A. CS235
- B. CS111
- C. **CS110**

(2 points) Testing

Consider the following simple class.

```
class Container:
    def __init__(self, width, height):
        assert width > 0 and height > 0
        self.height = height
        self.width = width

    def in_bounds(self, x, y):
        if x < 0 or x >= self.width:
            return False
        if y < 0 or y >= self.height:
            return False
        return True
```

(21) (1 pt) Which of the following docstrings would **NOT** be a valid doctest for the in_bounds function?

A. `>>> c = Container(3,4)`
   `>>> c.in_bounds(3,4)`
   `True`

B. `>>> c = Container(3,3)`
   `>>> c.in_bounds(4,2)`
   `False`

C. `>>> c = Container(512,512)`
   `>>> c.in_bounds(0,0)`
   `True`

D. `>>> c = Container(5,25)`
   `>>> c.in_bounds(-1,0)`
   `False`

E. `>>> c = Container(13, 13)`
   `>>> c.in_bounds(11,11)`
   `True`

(22) (1 pt) Which of the following would be a good pytest function to verify that the in_bounds function properly responds to a negative y value?

A.
```
def test_negative_y_false():
    c = Container(2, 2)
    assert c.in_bounds(-2,1) == False
```

B.
```
def test_negative_y_false():
    c = Container(2, 2)
    assert c.in_bounds(2,-1) == False
```

C.
```
def test_negative_y_false():
    c = Container(2, 2)
    assert c.in_bounds(1,-1) == False
```

D.
```
def test_negative_y_false():
    c = Container(2, 2)
    assert c.in_bounds(-1,-1) == False
```

E.
```
def test_negative_y_false():
    c = Container(2, 2)
    assert c.in_bounds(0,0) == True
```

(10 points) Language Parsing - Postfix Notation

In the calculator project, you parsed the input expressions to create a syntax tree, and then evaluated that tree to find the value of the expression that was provided. However, once we have the syntax tree, we can do more than just evaluate the expression. For example, we could use that syntax tree to rewrite the expression into another language or syntax. For example if we had a syntax tree of a python program, it would be possible to rewrite that program in C++ or Java just from the syntax tree (assuming there were translations for everything).

In this question, we are going to take our calculator expressions, which were written in what is known as prefix notation, where the operator precedes the operands, and convert them to postfix notation, where the operands precede the operator. Some examples:

If we had the following calculator expression for 1 + 2:
(+ 1 2)
It would be written in postfix notation as
1 2 +

The expression for (7 * 6) - (1 + 4):
(- (* 7 6) (+ 1 4))
Would be written as
7 6 * 1 4 + -

Notice that the postfix notation doesn't require any parentheses. When evaluating, each operator simply uses the two operands before it and replaces operands and operator with the value of the operation at its position in the expression as it is evaluated. Thus that last expression would evaluate like this:

7 6 * 1 4 + -
42 1 4 + -
42 5 -
37

> (23) (1 pt) What would the postfix notation be for the expression (* 3 2)
>   A. 3 * 2
>   B. * 2 3
>   C. 3 2 *
>   D. 2 3 *
>   E. (3 2 *)

> (24) (1 pt) What would the postfix notation be for the expression (+ 3 (* 2 1))
>   A. (1 * 2) + 3
>   B. ((2 1 *) 3 +)
>   C. 3 + (2 * 1)
>   D. 3 2 1 * +
>   E. 3 + (2 1 *)

Let's build a function that can traverse our calculator syntax tree and write out the expression in postfix notation. Remember that our syntax tree is composed of Pair objects where *first* is either an operator or a Pair object containing a sub expression and *rest* is either the next Pair object in the sub-expression or nil which represents the end of that expression. Thus our calculator expression (+ (* 3 2) 1) would be represented as:

```
Pair('+', Pair(Pair('*', Pair(3, Pair(2, nil))), Pair(1, nil)))
```
and would be written in postfix notation as:
```
3 2 * 1 +
```

Here's the framework for the code to write this out

```
def postfix(syntax_tree):
    """
    tree is a calculator syntax tree represented by Pair objects.  This
    function returns a string representing the calculator expression in
    postfix notation.  E.g. the calculator expression (+ (* 3 2) 1) would
    be represented by the string '3 2 * 1 +'.
    >>> postfix(Pair('+',Pair(Pair('*',Pair(3,Pair(2, nil))),Pair(1,nil))))
    '3 2 * 1 + '
    """
    if syntax_tree is nil:
        _____(a)_____
    if isinstance(syntax_tree, (float, int)):
        _____(b)_____
    if isinstance(syntax_tree, Pair):
        if _____(c)_____:
            return postfix(__(d)__) + postfix(__(e)__) + f"{__(f)__} "
        else:
            return postfix(syntax_tree.first)
```

Use the syntax tree for (+ 1 (* 3 2)) (given in the docstring) to help you figure out the code that should fill in the blanks and answer the following questions:

(25) (2 pts) What is/are the base case(s) for this recursive function? Select all that apply
    A. syntax_tree is a Pair object
    B. syntax_tree is the nil object
    C. syntax_tree is one of the operators (+, -, *, /)
    D. syntax_tree is a floating point number or an integer
    E. syntax_tree is something else

(26) (1 pt) What should be returned in the case where syntax tree is nil? What code should go in line (a)?
    A. return syntax_tree
    B. return ""
    C. return None
    D. Return 0

(27) (1 pt) What should be returned in the case where t is a floating point or integer number? What code should go in line (b)?

A. `return 0`

B. `return None`

C. `return t`

D. `return f"{syntax_tree}"`

E. `return f"{syntax_tree} "`

(28) (1 pt) How do we check to see if the item in the current pair is one of our operators? What code should go in line (c)?

A. `syntax_tree in ['+', '-', '*', '/']`

B. `syntax_tree in [+, -, *, /]`

C. `syntax_tree.first in ['+', '-', '*', '/']`

D. `syntax_tree .first in [+, -, *, /]`

If the current item is one of our operators, we need to return the first operand, the second operand, and then the operator. The following questions ask you to fill in the items for lines (d), (e), * (f) to make that happen.

(29) (1 pt) What value should go in line (d)?

A. `0`

B. `nil`

C. `None`

D. `tree`

E. `syntax_tree.first`

F. `syntax_tree.rest`

G. `syntax_tree.rest.rest`

(30) (1 pt) What value should go in line (e)?

A. `0`

B. `nil`

C. `None`

D. `tree`

E. `syntax_tree.first`

F. `syntax_tree.rest`

G. `syntax_tree.rest.rest`

(31) (1 pt) What value should go in line (f)?

A. `0`

B. `nil`

C. `None`

D. `syntax_tree`

E. `syntax_tree.first`

F. `syntax_tree.rest`

G. `syntax_tree.rest.rest`