CS 61A Structure and Interpretation of Computer Programs
Summer 2016

Final

INSTRUCTIONS

- You have 2 hours and 50 minutes to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except two 8.5" × 11" cheat sheets of your own creation.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
Instructional account (`cs61a-_`)	
BearFacts email (`_@berkeley.edu`)	
TA	
Name of the person to your left	
Name of the person to your right	
All the work on this exam is my own. **(please sign)**	

1. (6 points) Does Jack Like Jackfruits?

For each of the statements below, write the output displayed by the interactive Python interpreter when the statement is executed. The output may have multiple lines. **No answer requires more than three lines.** If executing a statement results in an error, write 'Error', but include all lines displayed before the error occurs. The first two have been provided as examples.

Assume that you have started `python3` and executed the following statements:

```
class Fruit:
    ripe = False
    def __init__(self, taste, size):
        self.taste = taste
        self.size = size
        self.ripe = True
    def eat(self, eater):
        print(eater, 'eats the', self.name)
        if not self.ripe:
            print('But it is not ripe!')
        else:
            print('What a', self.taste, 'and', self.size, 'fruit!')

class Tomato(Fruit):
    name = 'tomato'
    def eat(self, eater):
        print('Adding some sugar first')
        self.taste = 'sweet'
        Fruit.eat(self, eater)

mystery = Fruit('tart', 'small')
tommy = Tomato('plain', 'normal')
```

```
>>> mystery.taste
'tart'
```

```
>>> mystery.name
Error
```

```
>>> mystery.ripe
```

```
>>> tommy.eat('Brian')
```

```
>>> Tomato.ripe
```

```
>>> Tomato.eat(mystery, 'Marvin')
```

```
>>> Fruit.eat(tommy, 'Brian')
```

```
>>> tommy.name = 'sweet tomato'
>>> Fruit.eat = lambda self, own: print(
...                   self.name, 'is too sweet!')
>>> tommy.eat('Marvin')
```

2. (12 points) Marsalists Have Mutant Powers

(a) **(6 pt)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

A complete answer will:

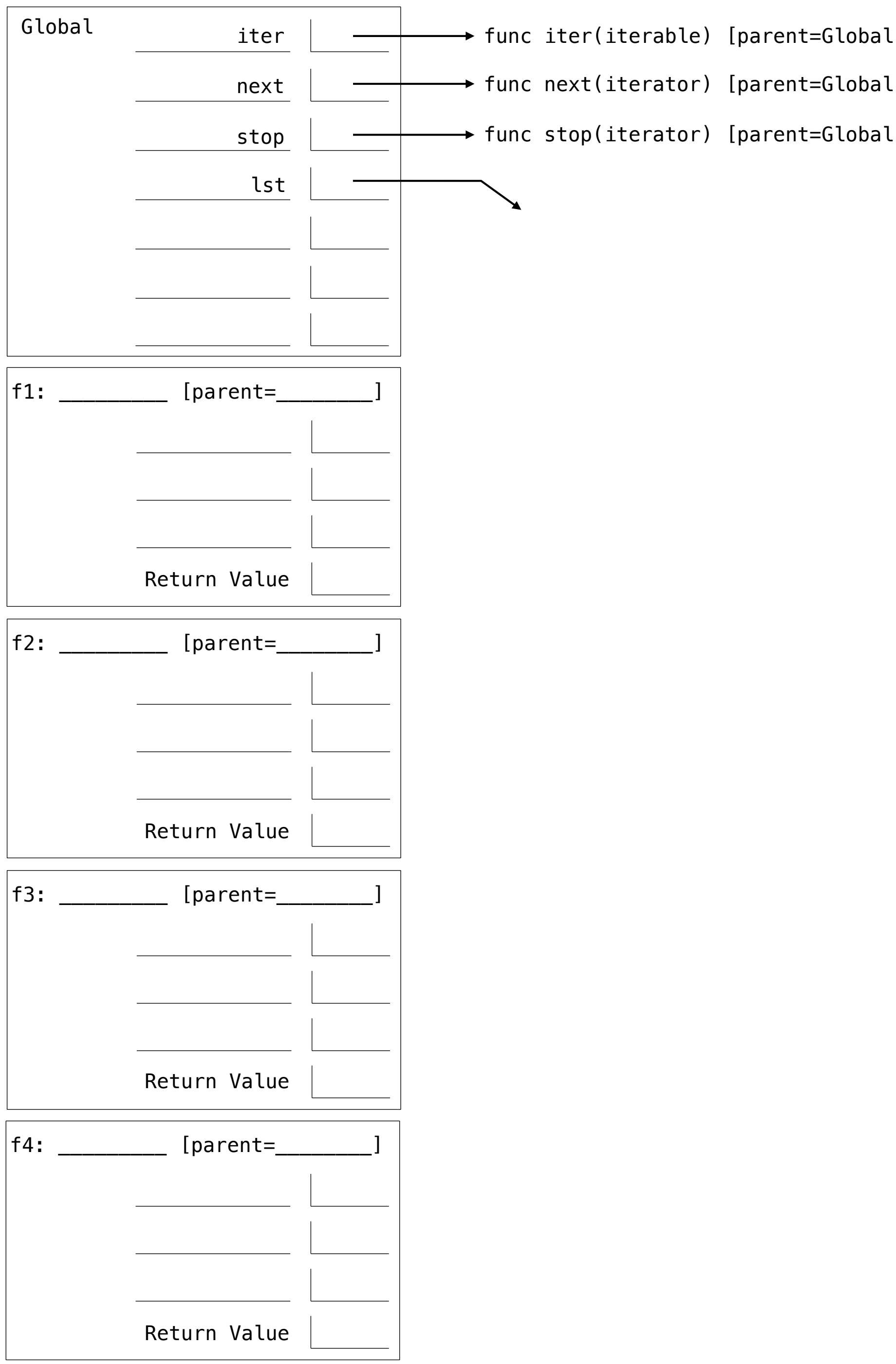
- Add all missing names and parent annotations to all frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
def iter(iterable):
    def iterator(msg):
        nonlocal iterable
        if msg == 'next':
            next = iterable[0]
            iterable = iterable[1:]
            return next
        elif msg == 'stop':
            raise StopIteration
    return iterator

def next(iterator):
    return iterator('next')

def stop(iterator):
    iterator('stop')

lst = [1, 2, 3]
iterator = iter(lst)
elem = next(iterator)
```



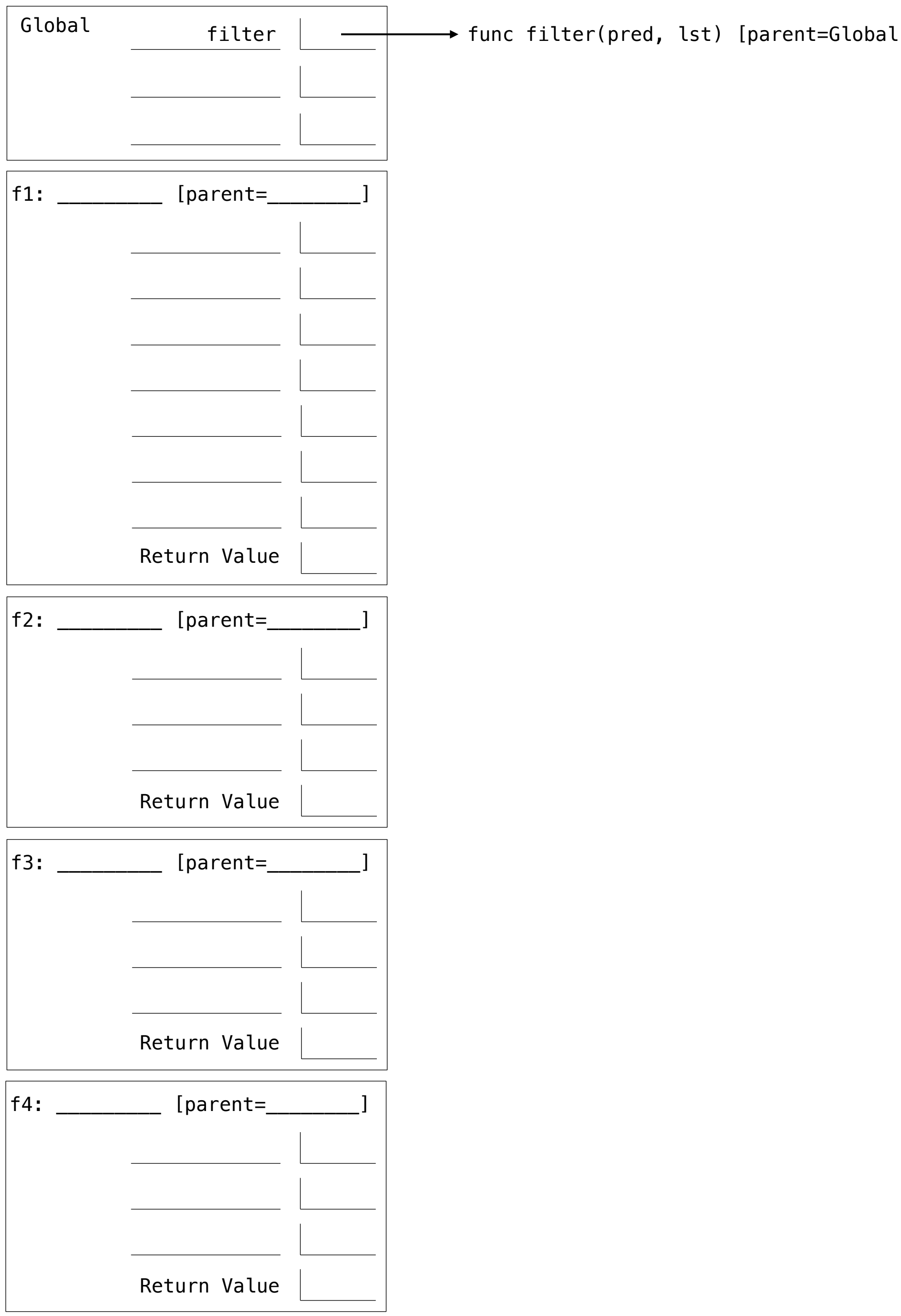
(b) **(6 pt)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

A complete answer will:

- Add all missing names and parent annotations to all frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
def filter(pred, lst):
    i = 0
    while i < len(lst):
        elem = lst[i]
        if not pred(lst[i]):
            elem = 'X'
        i += 1
    j = 0
    while j < len(lst):
        if lst[j] == 'X':
            lst.pop(j)
        else:
            j += 1

filter(lambda x: x < 2, [1, 2, 3])
```



3. (8 points) The Pair of Pair-ic and Pair-ome

(a) **(6 pt)** Define the function `combine_pairs` that takes a two-argument function `func` and returns a new function that takes in a list `lst`. When called, the new function should combine the elements in `lst` two by two using `func`, and return a new list of the results. If the length of `lst` is odd, the last element should be included in the return value without being combined with anything. See the doctests for details.

You may only use the lines provided. You may not need to fill all the lines.

```
def combine_pairs(func):
    """
    >>> from operator import add
    >>> add_pairs = combine_pairs(add)
    >>> add_pairs([1, 2, 3, 4])
    [3, 7]
    >>> add_pairs([1, 2, 3, 4, 5])
    [3, 7, 5]
    >>> group_pairs = combine_pairs(lambda x, y: [x, y])
    >>> group_pairs([‘hi’, ‘there’, ‘bye’, ‘now’])
    [[‘hi’, ‘there’], [‘bye’, ‘now’]]
    """

    def pairer(lst):

        result = _______________________________________________________

        for i in _______________________________________________________

            try:

                ________________________________________________________

            except IndexError:

                ________________________________________________________

        return _________________________________________________________

    return _____________________________________________________________
```

(b) **(2 pt)** Assuming the function `combine_pairs` works correctly, fill in the following line of code to correctly define the function `sum`, which takes in a non-empty list of numbers and returns the sum of the numbers. **Use only one line of code. If you need more space, you can continue your line of code on the second blank.** The `add` function has been imported for you; you may find it to be useful.

Hint: You may need the ternary operator `<expr1> if <cond> else <expr2>`.

```
>>> from operator import add

>>> sum = ______________________________________________________________

          ______________________________________________________________

>>> sum([1, 2, 3])
6
>>> sum([2, 0, 1, 6])
9
```

4. (9 points) Caught Ya!

Implement the function `catch_up`, which takes in two linked lists of integers `lnk1` and `lnk2` and mutates the linked list with the lower sum by repeatedly inserting 1 at the end until the sums are equal. See the doctests for details. **You may assume that the two linked lists that are passed in are non-empty and have the same length.** The `Link` class is provided for your reference.

You may only use the lines provided. You may not need to fill all the lines. You may not use methods that are not implemented in the `Link` class below.

Hint: You may need the ternary operator `<expr1> if <cond> else <expr2>`.

```
def catch_up(lnk1, lnk2):
    """
    >>> odds = Link(1, Link(3, Link(5, Link(7))))
    >>> evens = Link(2, Link(4, Link(6, Link(8))))
    >>> catch_up(odds, evens)
    >>> print(odds)  # odds is mutated
    <1 3 5 7 1 1 1 1>
    >>> print(evens)
    <2 4 6 8>
    """
    def catcher(lnk1, lnk2, sum1, sum2):

        sum1 = ________________________________________________________

        sum2 = ________________________________________________________

        if ____________________________________________________________

            lower = ___________________________________________________

            for _______________________________________________________

                _______________________________________________________

                _______________________________________________________

                _______________________________________________________

        else:

            catcher(__________________________________________________)

    catcher(__________________________________________________________)

class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first, self.rest = first, rest

    def __str__(self):
        string = '<'
        while self.rest is not Link.empty:
            string += str(self.first) + ' '
            self = self.rest
        return string + str(self.first) + '>'
```

5. (9 points) The Tree-o of Tam-tree, Tree-tas, and Tree-il

(a) (3 pt) Define the function `min_leaf_depth`, which takes in a tree `t` and returns the minimum depth of any of the leaves in `t`. Recall that the depth of a node is defined as how far away that node is from the root. See the doctests for details. The `Tree` class is provided for your reference.

You may only use the lines provided. You may not need to fill all the lines. You may not use methods that are not implemented in the `Tree` class below.

Hint: You may find the built-in `min` function useful.

```
def min_leaf_depth(t):
    """
    >>> t1 = Tree(2)
    >>> min_leaf_depth(t1)
    0
    >>> t2 = Tree(2, [Tree(0), Tree(1), Tree(6)])
    >>> min_leaf_depth(t2)
    1
    >>> t3 = Tree(2, [Tree(0), t2])
    >>> min_leaf_depth(t3)
    1
    >>> t4 = Tree(2, [t2, t3])
    >>> min_leaf_depth(t4)
    2
    """

    if t.is_leaf():

        ----------------------------------------------------------------

    else:

        ----------------------------------------------------------------

        ----------------------------------------------------------------


class Tree:
    def __init__(self, entry, children=[]):
        for c in children:
            assert isinstance(c, Tree)
        self.entry = entry
        self.children = children

    def is_leaf(self):
        return not self.children
```

(b) **(6 pt)** Define the function `find_path`, which takes in a binary search tree `bst` and a number `n` and returns a list representing the path through `bst` starting from the root and ending at `n`. You may assume that `n` exists in `bst` exactly once. See the doctests for details. The `BST` class is provided for your reference.

You may only use the lines provided. You may not need to fill all the lines. You may not use methods that are not implemented in the `BST` class below.

```
def find_path(bst, n):
    """
    >>> bst = BST(4, BST(2, BST(1)), BST(5))
    >>> find_path(bst, 1)
    [4, 2, 1]
    >>> find_path(bst, 2)
    [4, 2]
    """

    if _________________________________________________________________

        ________________________________________________________________

    elif ______________________________________________________________

        ________________________________________________________________

    else:

        ________________________________________________________________

class BST:
    empty = ()
    def __init__(self, entry, left=empty, right=empty):
        assert left is BST.empty or isinstance(left, BST)
        assert right is BST.empty or isinstance(right, BST)

        self.entry = entry
        self.left, self.right = left, right

        if left is not BST.empty:
            assert left.max <= entry
        if right is not BST.empty:
            assert entry < right.min

    @property
    def max(self): # Returns the maximum element in the tree
        if self.right is BST.empty:
            return self.entry
        return self.right.max

    @property
    def min(self): # Returns the minimum element in the tree
        if self.left is BST.empty:
            return self.entry
        return self.left.min
```

6. (11 points) Dan-scheme with the Cars

(a) **(3 pt)** Define the procedure `digit-prod`, which takes in a non-negative integer `n` and returns the product of the digits of `n`. See the example usage for more details.

You may only use the lines provided. You may not need to fill all the lines.

Hint: The built-in `quotient` and `modulo` procedures return the quotient and remainder, respectively, of one number divided by another.

```
(define (digit-prod n)

  ----------------------------------------------------------------------

  ----------------------------------------------------------------------

  ---------------------------------------------------------------------)

scm> (quotient 5 3)
1

scm> (modulo 5 3)
2

scm> (digit-prod 12345)
120

scm> (digit-prod 200)
0

scm> (digit-prod 2222)
16
```

(b) **(5 pt)** Define the procedure `merge`, which takes in two sorted lists of numbers `lst1` and `lst2` and returns a new list that contains all the elements in the two lists in sorted order. See the example usage for more details.

You may only use the lines provided. You may not need to fill all the lines.

Hint: The built-in `append` procedure may be useful.

```
(define (merge lst1 lst2)

  (cond ----------------------------------------------------------------

        ----------------------------------------------------------------

        ----------------------------------------------------------------

        --------------------------------------------------------------))


scm> (merge ‘(1 3 5) ‘(2 4 6))
(1 2 3 4 5 6)

scm> (merge ‘() ‘(2 4 6))
(2 4 6)

scm> (merge ‘(5 7) ‘(2 4 6))
(2 4 5 6 7)
```

(c) **(3 pt)** For each of the following Scheme expressions, circle the correct number of calls that would be made to `scheme_eval` and `scheme_apply` when evaluating the expression in our Scheme interpreter (Project 4). Assume you are **not** using the tail-recursive `scheme_optimized_eval`.

```
(- (* 6 11) 2 2 2)
```

Number of calls to `scheme_eval`: 1 4 7 9 12

Number of calls to `scheme_apply`: 0 1 2 3 4

```
(if (+ 0 (- 1 1)) (+ 4 5) 4)
```

Number of calls to `scheme_eval`: 1 4 7 9 12

Number of calls to `scheme_apply`: 0 1 2 3 4

```
((lambda (x) (* x x)) 57)
```

Number of calls to `scheme_eval`: 1 4 7 9 12

Number of calls to `scheme_apply`: 0 1 2 3 4

7. (8 points) Let to Sam-da

In the Scheme project, you wrote a procedure `let-to-lambda` that took in a Scheme expression as a list and transformed all `let` expressions within it into equivalent `lambda` expressions. We will now do the same thing in Logic. Define the incomplete facts below to complete the `let-to-lambda` relation, which relates a Scheme expression to the equivalent expression with no `let` expressions. See the example usage on the next page for more details.

Symbolic programming is very powerful in Logic. Our `let-to-lambda` will work the exact same way as the `let-to-lambda` procedure in Scheme, with the tiny exception that we cannot use `let` as a variable name.

You may only use the lines provided. You may not need to fill all the lines.

Facts for the `equal`, `is-pair`, and `zip` relations have been defined for you, and you may find these relations useful. See the example usage for more details. **You may not assume that any other relations have been defined.**

```
(fact (equal ?x ?x))

(fact (is-pair (?car . ?cdr)))

(fact (zip () () ()))
(fact (zip (?f1 . ?r1) (?f2 . ?r2) ((?f1 ?f2) . ?r-zip))
      (zip ?r1 ?r2 ?r-zip))


logic> (query (is-pair (1 2 3)))
Success!

logic> (query (is-pair 1))
Failed.

logic> (query (zip ?lst1 ?lst2 ((a 1) (b 2))))
Success!
lst1: (a b)     lst2: (1 2)
```

```
; atoms (expressions that are not pairs) do not need conversion

(fact (let-to-lambda ?x ?x)

      ________________________________________________________________)


; quoted expressions, even if they contain let, should not be converted

(fact (let-to-lambda (quote ?expr) (quote ?expr)))


; lets in the body of lambda expressions should be converted

(fact (let-to-lambda (lambda ?formals . ?body) (lambda ?formals . ?body-conv))

      (let-to-lambda ______________________________________________))
```

```
; lets in the body of define expressions should be converted

(fact (let-to-lambda (define ?formals . ?body) (define ?formals . ?body-conv))

      (let-to-lambda ?body ?body-conv))


; lets like (let ((a 1)) a) should be converted into ((lambda (a) a) 1)
; remember that the bodies of lets can contain more lets
; remember that the arguments in let bindings can also contain more lets

(fact (let-to-lambda __________________________________________________)

      (zip ____________________________________________________________)

      (let-to-lambda __________________________________________________)

      (let-to-lambda _________________________________________________))


; check and convert pairs not in one of the above forms

(fact (let-to-lambda (?car1 . ?cdr1) (?car2 . ?cdr2))

      (not (equal ?car1 quote))

      (not (equal ?car1 lambda))

      (not (equal ?car1 define))

      (not (equal ?car1 let))

      (let-to-lambda ?car1 ?car2)

      (let-to-lambda ?cdr1 ?cdr2))


logic> (query (let-to-lambda (+ 1 2) ?conv))
Success!
conv: (+ 1 2)

logic> (query (let-to-lambda (let ((a 1) (b 2)) (+ a b)) ?conv))
Success!
conv: ((lambda (a b) (+ a b)) 1 2)

logic> (query (let-to-lambda (let ((a (let ((a 2)) a)) (b 2)) (+ a b)) ?conv))
Success!
conv: ((lambda (a b) (+ a b)) ((lambda (a) a) 2) 2)
```

8. (5 points) Zhen-erators Produce Power

Implement the generator function `powers_of_two` that iterates over the infinite sequence of non-negative integer powers of two, starting from 1. You must do this by selectively including elements from an infinite sequence of integers, created by calling the provided `integers` generator function.

You may only use the lines provided. You may not need to fill all the lines.

Hint: You may find the `drop` generator function useful.

```
def integers(n):
    while True:
        yield n
        n += 1

def drop(n, s):
    for _ in range(n):
        next(s)
    for elem in s:
        yield elem


def powers_of_two(ints=integers(_________)):
    """
    >>> p = powers_of_two()
    >>> [next(p) for _ in range(10)]
    [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
    """

    curr = ____________________________________________________________

    yield ______________________________________________________________

    yield from ________________________________________________________
```

9. (2 points) Assorted Trivia-n

For the following four questions, choose two or more to answer. Each correct answer you provide is worth one point, but you can only earn a maximum of two points.

(a) (1 pt) What is one of the benefits that Apache Spark provides?

(b) (1 pt) In one sentence, describe a problem other than the halting problem that is undecidable.

(c) (1 pt) In one sentence, describe one reason why the Caesar cipher can be easily broken by a computer.

(d) (1 pt) In one sentence, what is a rollout in the context of reinforcement learning?

10. (0 points) We Must All Brain Together

In the box below, write a positive integer. The student who writes the lowest unique integer will receive one extra credit point. In other words, write the smallest positive integer that you think no one else will write.

However, if no one's answer is unique, then everyone receives one extra credit point. So an optimal solution would be, for example, if everyone wrote 1. Do you trust your fellow students?

11. (0 points) What a Marvelous Summer

The staff have used the letters of this exam to encode a message for you! Thank you for a fantastic semester. Write the decoded message in the box below. (This isn't worth any extra credit.)

```
4[0]  11[11]  6[2]  4[3]  2[2]  3[4]  10[5]  4[2]  1[8]  9[0]  5[3]  3[5]  1[1]  8[3]  7[5]  4[-1]
```