Midterm Exam

Structure and Interpretation of Computer Programs
Department of Computer Science and Technology, Nanjing University

10:10 - 12:10, November 10, 2021

Instructions

- The exam is closed book (except dictionaries), closed notes, closed computer, closed calculator.
- You have 2 hours to complete the exam. Your answer could be either in English or in Chinese.
- Mark your answers **on the exam itself**. We will not grade answers written on scratch paper.

Policies & Clarifications

- You may use built-in Python functions that do not require import, such as `min`, `max`, `pow`, `len`, `abs`, `sum`, `next`, `iter`, `list`, `tuple`, `map`, `filter`, `zip`, `all`, and `any`.
- For fill-in-the blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.

Student ID	
Student Name (in Chinese)	

Do NOT open the exam paper until the exam has started.

For staff use only

Problem	1	2	3	4	5	6	Extra	Total
Score								

This page is intentionally left blank.

1 Book Me on Weekdays (11pts)

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is valued. The output may have multiple lines. Each expression has at least one line of output.

- If an error occurs, write **`Error`**, but include all output displayed before the error.
- To display a function value, write **`Function`**. To display an iterator value, write **`Iterator`**.
- If an expression would take forever to evaluate, write **`Forever`**.

The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

Assume that you have first started `python3` and executed the statements on the left.

```
weekday = lambda d: not weekend(d)
weekend = lambda d: \
            d % 7 == 6 or d % 7 == 0
luckday = lambda d, l: d % l == 0

today = 20211110

def future(today):
    if weekend(today):
        print('weekend')
    else:
        yield today
    yield from future(today + 1)

def appoint(dates, cond, act):
    for date in dates:
        if cond(date):
            act = act(date)

def booking(n=0):
    def booked(d):
        print(n + 1, d)
        return booking(n + 1)
    return ok if n >= 3 else booked

ok = lambda ok: booking(ok + 1)
```

Expression	Interactive Output
`pow(2, 11) - 27`	`2021`
`print(4, 5) + 1`	`4 5` `Error`
`print(print(2), 4)`	
`len([1, 2, 3, [4, 5]])`	
`range(10)[2]`	
`today is weekday or weekend`	
`luckday(today)(2)`	

`a = [0, 1]`
`b = iter(a)`
`for x in a:`
`  for y in b:`
`  print(x, y)`	

`f = future(4)`
`for i in range(3):`
`  print(next(f))`	

`x = [1, 2, 3, 4, 5, 6, 7]`
`y = weekday`
`z = booking`
`appoint(x, y, z)`	

2 A Tale of Two Diagrams (26pts)

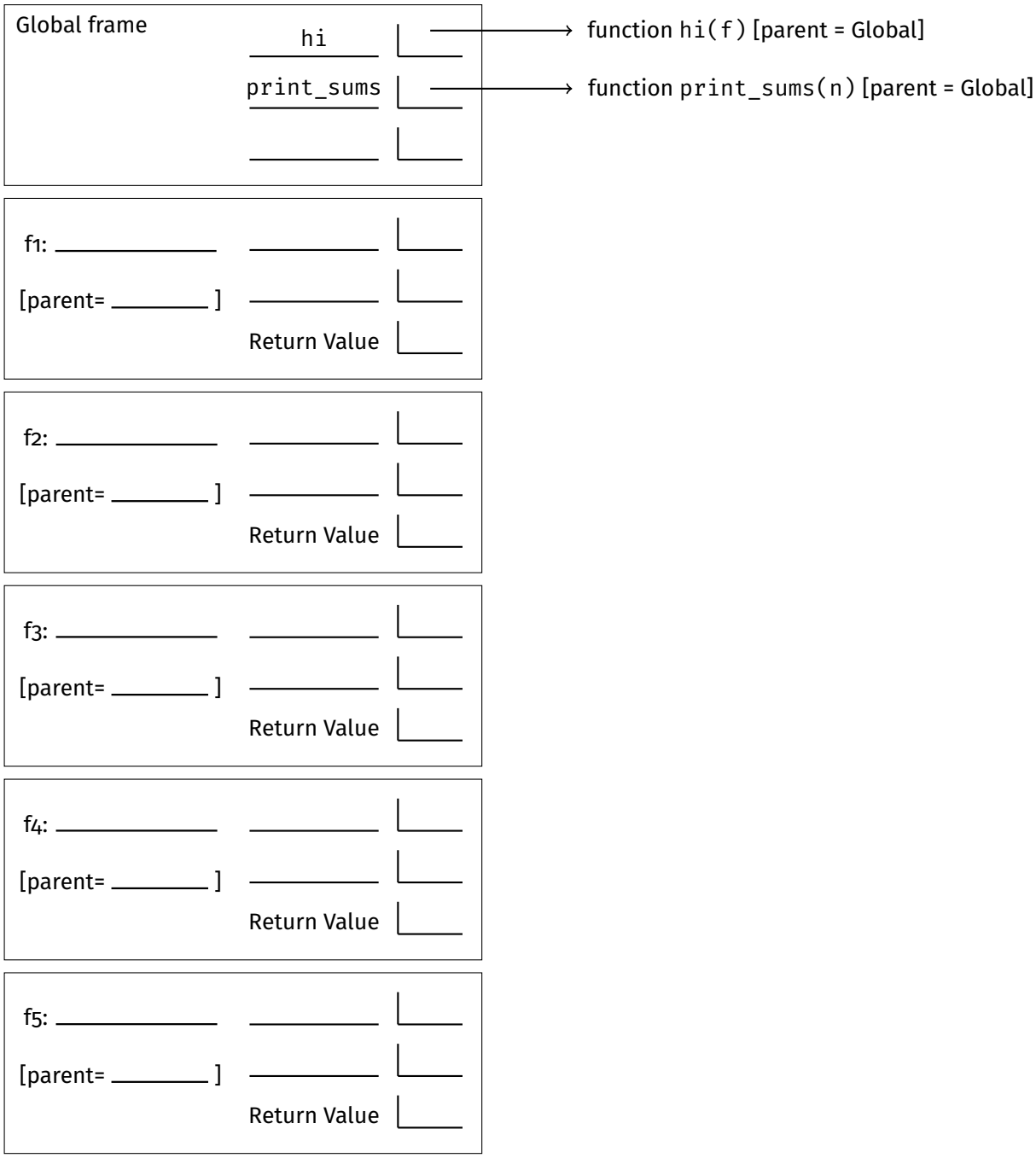
2.1 Frames (14pts)

Fill in the environment diagram that results from executing the code on the right 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 missing names and parents to all local frames.
- Add missing values created or referenced during execution, and show the return value for each local frame.

```
1 def hi(f):
2     def mut(s):
3         nonlocal f
4         f = f(s)
5         return f
6     return mut
7
8 def print_sums(n):
9     print(n)
10    return lambda k: print_sums(n + k)
11
12 st = hi(print_sums)(2020)(1)
```



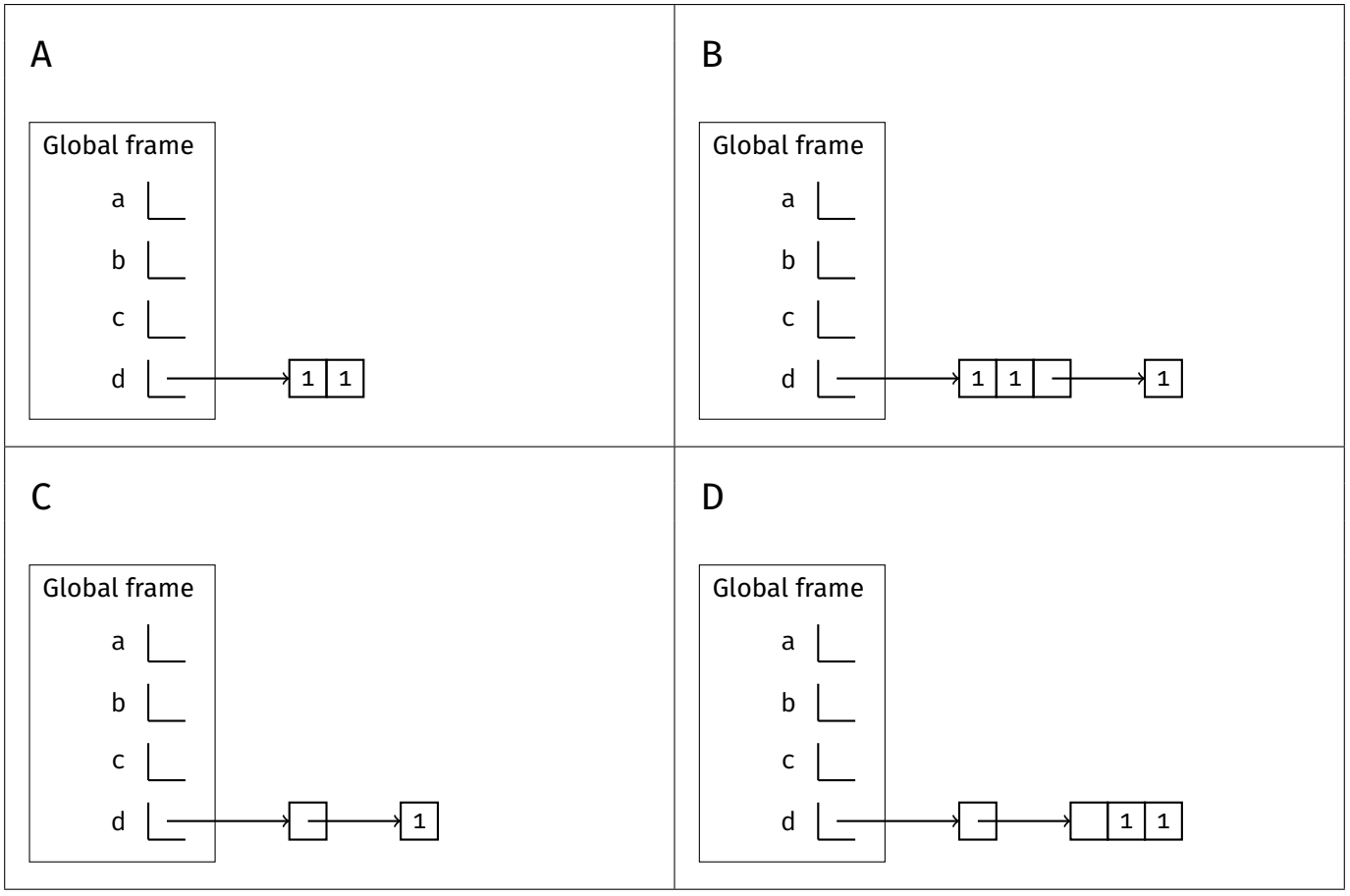
2.2 Boxes (12pts)

Read the following program carefully, and decide which environment diagram would result **after** executing each line of code. Write your choice in each box next to a line of code, then complete the corresponding diagram by adding all missing values (boxes and arrows). You only need to complete three chosen diagrams and left the remaining one untouched.

(1) Fill in the boxes with A, B, C, or D

```
a = [1, 1]
b = [1, 1]
c = [a] + b
d = c[1:]

while b:
    a.append([b.pop()])
    a, d = d, a
b = a

a[2][0], d = c, a[2]
```

(2) Complete the chosen diagrams



3 Subsequence (15pts)

We say that a list `s` is a **subsequence** of a list `l`, if all values in `s` occur in `l` in the same order, but not necessarily one after the other. Otherwise, `s` is not a subsequence of `l`. Write a predicate function `subseq` that takes two list `l` and `s` as arguments, and determines whether `s` is a subsequence of `l`. If so, the function should return `True`; otherwise, it should return `False`. We have provided a few doctests to demonstrate its definition and usage.

```
def subseq(l, s):
    """
    >>> subseq([1, 2, 3, 4, 5, 6], [2, 4, 6])
    True
    >>> subseq([6, 5, 4, 3, 2, 1], [3, 4])
    False
    >>> subseq([1, 9, 2, 8, 3, 7], [1, 2, 3, 4])
    False
    >>> subseq([2, 3, 5, 7, 11, 13], [2, 2, 3])
    False
    """

    if s == []:

        return ____________________________________________________________

    elif l == []:

        return ____________________________________________________________

    elif ____________________________________________________________:

        return subseq(_____________________________, _____________________________)

    return ____________________________________________________________
```

We can use an iterator to represent unvisited elements. Implement this alternative version of `subseq`.

Hint: The Python built-in function `all(iterable)` returns `True` if `bool(x)` is `True` for all values `x` in the iterable.

```
def subseq(l, s):

    def helper(it, target):

        for value in it:

            if ____________________________________________________________:

                return True

        return False

    it = iter(_______________)

    return all([_____________________________________ for v in _______________])
```

4 Announce Highest Revisited (10pts)

Implement announce, which is a higher order function that can be called repeatedly and detects highest integer arguments. As a side effect of repeated calls, it prints each argument that is strictly larger than all previous ones. **You may assume the initial highest is set to zero.** A few doctests are provided to clarify its definition and usage.

```
def announce(n):
    """
    >>> f = announce(0)(1)(2)(1)(2)(3)(4)(1)
    1
    2
    3
    4
    >>> g = announce(-1)(0)(5)(2)(6)(10000)(888)
    5
    6
    10000
    >>> g = g(10000)(10086)(10001)
    10086
    """

    return ____________________________________________________________(n)

def detect(max_value):

    def helper(cur_value):

        if ____________________________________________________________:

            ____________________________________________________________

        return ____________________________________________________________

    return helper
```

Now, assume that you have implemented announce correctly, write a function that prints an increasing subsequence of list *l*. You may assume that all elements in *l* are positive integers. Definition of subsequence is in problem 3.

```
def increasing_subsequence(l):
    """
    >>> increasing_subsequence([1, 2, 2, 4, 3, 5])
    1
    2
    4
    5
    """

    f = ____________________________________________________________

    while ____________________________________________________________:

        f, l = _____________________________, _____________________________
```

Draw something here if you want! (optional, opts)

5 Composite Functions (16pts)

Haruki wants to implement a function called `composite`, which takes a positive integer `n` indicating the number of functions. The returned function takes one function at a time. After calling it for `n` times, it returns the composition of the functions in the order we pass them.

You may assume all the functions we input exactly takes one argument and returns one value. For example, `composite(n)(f1)(f2)...(fn)` should have same effect as `lambda x: f1(f2(...(fn(x)...))`.

5.1 Bugs Fixed (6pts)

Setsuna quickly implemented `composite`, but sadly it is a buggy one! Can you help her fix all the bugs?

```
 1 def composite_setsuna(n):
 2     """Composites N functions.
 3     >>> composite_setsuna(3)(lambda x: x + 2)(lambda x: x * 2)(lambda x: x - 2)(5)
 4     8
 5     >>> composite_setsuna(2)(lambda x: x ** 2)(lambda x: x // 2)(9)
 6     16
 7     """
 8     func = lambda x: x
 9     def helper(f):
10         if n < 0:
11             return func(f)
12         n -= 1
13         func = lambda x: func(f(x))
14         return helper
15     return helper
```

You can add a line by mentioning the position and content to insert, remove a line by mentioning its line number or edit a line by mentioning its line number with the content after editing.

For example, the table on the right means adding a line between line 20 and line 21 with content `p += 1`, removing line 22 and editing line 23 to `return p`.

Type	Line	Content
add	20-21	`p += 1`
remove	22	
edit	23	`return p`

Write your answer in the following table. **You may make at most 3 modifications.**

Type	Line	Content

5.2 List Comp-osited (5pts)

Haruki is greedy and he wants more. This time he wants to implement `composite_list`, which takes a non-empty list of functions and returns the composition of the functions in order.

Again, you may assume all the functions we input takes exactly one argument and returns one value. For example, `composite_list([f1, f2, ..., fn])` should have same effect as `lambda x: f1(f2(...(fn(x)...))`.

Setsuna realizes that `composite_list` is quite similar to `composite`. Therefore, it is a good idea to implement the new function using `composite_setsuna`! Assume all bugs in `composite_setsuna` have been fixed, your task is to implement `composite_list` with at most two lines of code. **If you use both lines, you will get at most 3 points.**

```
def composite_list(lst):
    """Composites all functions in LST.
    >>> composite_list([lambda x: x + 2, lambda x: x * 2, lambda x: x - 2])(5)
    8
    >>> composite_list([lambda x: x**2, lambda x: x // 2])(9)
    16
    """
    func = composite_setsuna(len(lst))

    ______________________________________________________________________

    ______________________________________________________________________

    return func
```

5.3 Difference Spotted (5pts)

Kazusa also finished implementing the `composite`, here is her answer.

```
def composite_kazusa(n):
    if n == 1:
        return lambda f: lambda x: f(x)
    return lambda f: lambda g: composite_kazusa(n - 1)(lambda x: f(g(x)))
```

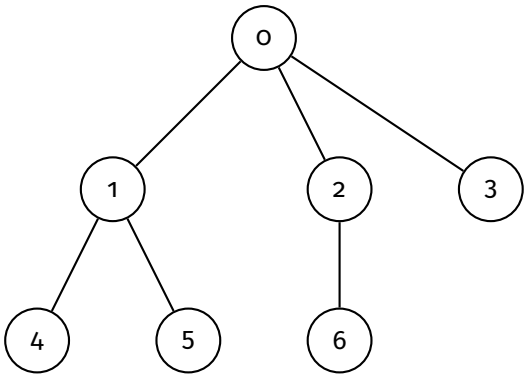
Although *Kazusa*'s answer looks different from *Setsuna*'s (again, assume all bugs have been fixed), they truly have same outputs with the inputs defined in doctests. However, subtle differences do exist when they deal with same inputs. Implement `tell_me_who`, which takes either `composite_setsuna` or `composite_kazusa` as the only argument and returns the name of the author of the input (i.e., 'Setsuna' or 'Kazusa'). **You may not use `__name__`, `repr`, `str` or any other way to obtain the name of functions.**

```
def tell_me_who(comp):
    """Returns the author of COMP.
    >>> tell_me_who(composite_setsuna)
    'Setsuna'
    >>> tell_me_who(composite_kazusa)
    'Kazusa'
    """

    ______________________________________________________________________

    ______________________________________________________________________

    return 'Setsuna' if ____________________________________________ else 'Kazusa'
```

6 Trees (22pts)

The following tree APIs are available for this problem:

- `tree(label, branches=[])`
- `label(tree)`
- `branches(tree)`
- `is_leaf(tree)`



6.1 Plant a Tree (3pts)

Define the tree in the figure using `tree` constructor.

```
my_tree =
```

6.2 Equivalent Trees (3pts)

We say two trees are **equivalent** if and only if they have the equivalent label and equivalent branches – even their branches are in different orders.

Meltryllis implemented `equals_tree`, which takes two trees as arguments, and returns `True` if the two trees are equivalent, otherwise `False`.

```
def equals_tree(t1, t2):
    """Return whether tree t1 equals to tree t2.
    >>> t1 = tree(1, [tree(2), tree(3)])
    >>> t2 = tree(1, [tree(3), tree(2)])
    >>> equals_tree(t1, t2)
    True
    """
    return label(t1) == label(t2) and \
         all([any([equals_tree(b1, b2) for b2 in branches(t2)]) for b1 in branches(t1)])
```

Is *Meltryllis*'s implementation correct? If you think it is right, give a brief reason in English or Chinese; otherwise, give an example of `t1` and `t2` that the above implementation will produce a wrong output.

6.3 Traverse Trees (6pts)

In previous assignments, we implemented `preorder` like this:

```
def preorder(t):
    """
    >>> preorder(my_tree)
    [0, 1, 4, 5, 2, 6, 3]
    """
    return sum([preorder(b) for b in branches(t)], [label(t)])
```

This time, we want to traverse a tree in level order (i.e. in order of node height), which means we will traverse each level of tree from left to right. We will start from root and never jump back to a upper level. For example, the level-order of `my_tree` is [0, 1, 2, 3, 4, 5, 6]. Implement `level_order`, which takes a tree `t` and yields the label of nodes in level order.

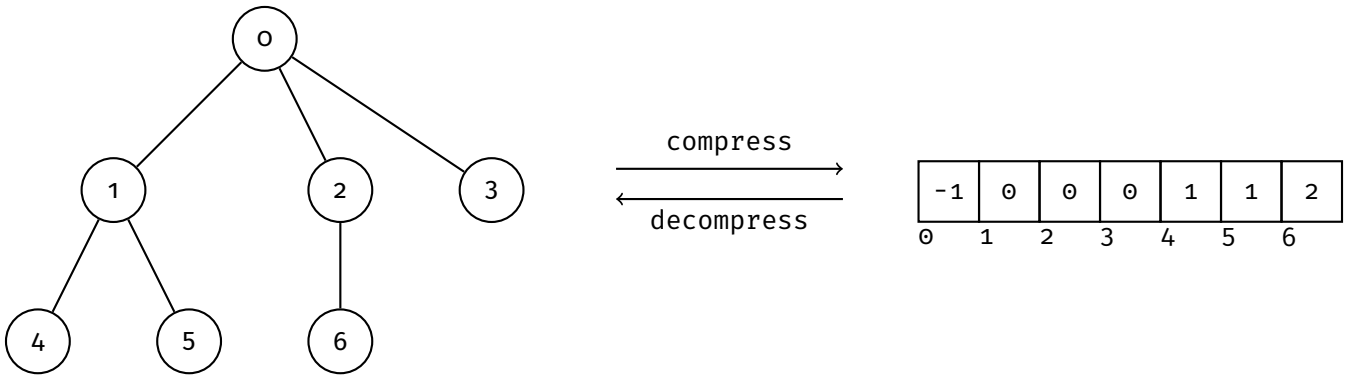
```
def level_order(t):
    """Generates all label of t, in level order.
    >>> list(level_order(my_tree))
    [0, 1, 2, 3, 4, 5, 6]
    """
    def helper(trees):

        if trees:

            yield from [____________________________ for t in ____________________________]

            yield from helper(_____________________________________________________________)

    yield from helper(_________________________________________________________________)
```

6.4 Compress Trees (10pts)

Although `preorder` and `level_order` can compress a tree to a list, it loses some information, which means it is hard to build the same tree, or even an equivalent tree from result of `preorder`. This time, we want a function to compress a tree into a list from which we can rebuild an equivalent tree.

For convenience, you may assume there are `n` nodes in the tree, labeled as 0, 1, . . . , `n` - 1, respectively. Please note that, labels can be different from level order, and the root of tree might not be labeled as 0.

We use a list of `n` integers to represent the compression. The index of an item in this list indicates the label of its corresponding node, and its content refers to the label of its parent node. Specially, content for an item representing the root node is -1. For example, the compression of `my_tree` is [-1, 0, 0, 0, 1, 1, 2], as shown in the following figure.



Implement `compress` and `decompress`, the former function takes a tree and returns the compressed result (i.e. a list of integers), the latter function takes a list and returns a tree that could be compressed into the same list.

```
def compress(n, t):
    """Returns the compressed result of TREE which have N nodes in total.
    >>> compress(7, my_tree)
    [-1, 0, 0, 0, 1, 1, 2]
    """

    result = [-1 for i in range(n)]

    def helper(tree, parent_label):

        ____________________________________________________________ = parent_label

        for branch in ____________________________________________________________:

            helper(_______________________________, _______________________________)

    helper(t, ______________________)

    return result

def decompress(lst):
    """Returns a tree equivalent to any tree that can be compressed into LST.
    >>> tree = decompress([-1, 0, 0, 0, 1, 1, 2])
    >>> equals_tree(tree, my_tree)
    True
    """

    def helper(current_label):

        return tree(_____________________, [____________________ for index \

                    in _____________________ if ___________________________________])

    return branches(_____________________)[0]
```

Extra Problems (3pts)

- The textbook for SICP is _____ .
- The legal name of teaching assistant *Meltryllis* is _____ (in Chinese or English).
- *Let's witness the wisdom of the crowd!*

  In this extra problem, you are asked to vote for **zero or one** teaching assistants (TAs). For each TA, all students will receive 1/5 (0.2) points if at least 15% of students vote for him. Theoretically, if votes are evenly distributed, everyone will gain 1 extra point! However, if you vote for more than one TA, you will get no extra points.

  Example: ☑ TA Vote: ☐ *Hengjie Chen* ☐ *Xuyang Li* ☐ *Tianyun Zhang* ☐ *Ziyang Yan* ☐ *Meltryllis*