CS 61A Structure and Interpretation of Computer Programs
Spring 2026

FINAL

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

You can complete and submit these questions before the exam starts.

(a) What is your full name?

(b) What is your student ID number?

(c) What is your @berkeley.edu email address?

(d) Sign (or type) your name to confirm that all work on this exam will be your own. The penalty for academic misconduct on an exam is an F in the course.

1. (13.0 points) What's that I smell? Oh, it's potpourri

(a) (2.0 points) Penny For Your Thoughts

i. (1.0 pt) You are gluing sequences of pennies and dimes to a popsicle stick to make bookmarks. You don't want to go over budget, so you write the following code to count the number of distinct bookmark designs you can make for a bookmark that has `n` cents on it.

```
def count(n):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    else:
        return count(n-1) + count(n-10)
```

What is the order of growth of your code with respect to the input `n`?

◯ Constant ◯ Logarithmic ◯ Linear ◯ Quadratic ◯ Exponential

ii. (1.0 pt) The following code takes in `data` as a list.

```
def process(data):
    result = []
    for d in data:
        result.append(d)
        for i in range(1000):
                result.append(d*i)
    return result
```

What is the code's order of growth with respect to the number of elements in `data`?

Assume `append` takes constant time.

◯ Constant ◯ Logarithmic ◯ Linear ◯ Quadratic ◯ Exponential

(b) (3.0 points) You Get It From Your Father

Assume the following code has been executed.

```
class A:
    x, y = 10, 20
    def f(self):
        return self.x + self.y

class B(A):
    y = 30
    def __init__(self):
        self.x = 5

class C(B):
    def __init__(self):
        self.x = 0
        super().__init__()
        self.y = self.x + self.y
```

Write the output that would be displayed by evaluating each expression below. If an error occurs, write ERROR, but also write any output that is displayed before the error occurs.

i. (1.0 pt) `B().f()`

ii. (2.0 pt) `C().f()`

(c) (3.0 points) Almost Summer

Assume the following code has been executed.

```
def f(s):
    yield from s
    yield from s

it = f("Summer")

next(it)
next(it)
next(it)
next(it)
print(next(it))
```

i. (1.0 pt) What is printed by the `print` statement?

ii. (2.0 pt) How many more times can `next(it)` be called after the print statement *before* the StopIteration exception is raised?

Write one of the following:

- an integer answer e.g. `99`
- `StopIteration` if we've already hit the StopIteration exception
- `Infinitely many` if it can be called an infinite number of times

(d) (5.0 points) Staying In Touch

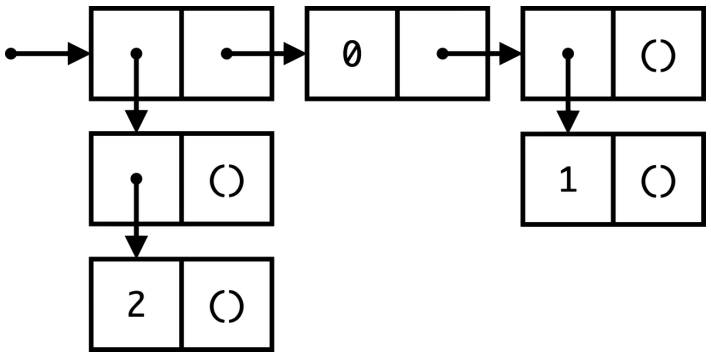
The `Contact` class stores a `name` and `email` for an address book entry. Two contacts can be merged into one by creating and returning a new `Contact` instance that keeps the first contact's email but combines names with `' & '`. Complete the following code to match the behavior of the doctests below.

```
class Contact:
    """
    >>> alice = Contact('Alice', 'alice@example.com')
    >>> alice
    Contact('Alice', 'alice@example.com')
    >>> print(alice)
    Alice <alice@example.com>
    >>> bob = Contact('Bob', 'bob@example.com')
    >>> alice.merge(bob)
    Contact('Alice & Bob', 'alice@example.com')
    >>> print(alice.merge(bob))
    Alice & Bob <alice@example.com>
    """
    def __init__(self, name, email):
        self.name = name
        self.email = email

    def __repr__(self):

        return ____________________________________________________

    def __str__(self):

        return ____________________________________________________

    def merge(self, other):

        return ____________________________________________________
```

2. (18.0 points) The Phoenician Scheme

A scheme list `s` has been defined.

```
scm> (draw s)
```



(a) (2.0 pt) Using only `s`, `(`, `)`, `CAR`, and `CDR`, write an expression to return `1`. Write `CAR` and `CDR` in uppercase.

```
scm> ____________________________________________________________________________________
```

(b) (2.0 pt) Using only numbers, `(`, `)`, `cons` (and `nil` if you need it), define `s`.

```
scm> (define s ____________________________________________________________________________

____________________________________________________________________________________________)
```

(c) (2.0 pt) Using only numbers, `(`, `)`, and `list`, define `s`. Do **not** use `quote` or the single quote character `'`.

```
scm> (define s ____________________________________________________________________________)
```

(d) (2.0 pt) Using only numbers, `(`, `)`, and `quote`, define `s`. Do **not** use the single quote character `'`.

```
scm> (define s ____________________________________________________________________________)
```

(e) (2.0 pt) Using only the number `1`, `(`, `)`, backquote `` ` ``(shorthand for quasiquote), comma `,` (shorthand for unquote), `+` and `-`, define `s`.

```
scm> (define s ____________________________________________________________________________)
```

(f) (4.0 pt) The `repeated` higher-order function takes a function `f` and a number `N` and returns a function `g(x)` that calls `f` on its input over and over `N` times. E.g., if `N` is `3`, `g(x) = f(f(f(x)))`. If `N` is `0`, `g(x)` just returns `x`. Fill in the blank to define `s`. Your solution must use `repeated` and process the input list `'(2 0 1)`.

```
scm> (define (plus1 x) (+ x 1))
scm> (define plus3 (repeated plus1 3))
scm> (plus3 10)
13

scm> (define s (____________________________________________________________ '(2 0 1) ))
```

(g) (4.0 pt) We want to define a Scheme macro that simulates Python's list comprehension:

```
>>> [10 * x for x in [2, 4, 1, 5] if x > 3]
[40, 50]
```

Thankfully, map and filter are built-in to Scheme:

```
(map <proc> <lst>)    ;; Returns a list constructed by calling proc (a one-argument procedure)
                         on each item in lst.
(filter <pred> <lst>) ;; Returns a list consisting of only the elements of lst that return
                         true when called on pred (a one-argument procedure).
```

Complete the following macro definition:

```
(define-macro (list-comprehension expr for sym in vals if pred)
    (list _______map|filter (list _______lambda (list _______sym) _______expr)
            (i)    (ii)            (iii)                (iv)        (v)

    `(_______map|filter (_______lambda (_______sym) _______pred) _______vals)
       (vi)   (vii)      (viii)           (ix)        (x)          (xi)
    )
)

scm> (list-comprehension (* 10 x) for x in '(2 4 1 5) if (> x 3))
(40 50)
```

For each `map|filter`, select the correct one. Then for each blank, select one of the following:

- `'` (normal forward quote)
- `,` (comma)
- `<leave it blank>`

i. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

ii. ◯ `map` ◯ `filter`

iii. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

iv. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

v. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

vi. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

vii. ◯ `map` ◯ `filter`

viii. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

ix. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

x. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

xi. ◯ `'` (quote) ◯ `,` (comma) ◯ `<blank>`

3. (16.0 points) An SQL Query walks over and says: "May I JOIN you?"

You are managing a music festival and have the following database. Any queries you write should also work if you are given larger versions of these tables.

artists:

aconst	name	genres	rec_labels
1	Sudan Archives	Electro, R&B	Stones Throw
2	Hozier	Blues, Folk, Soul	Rubyworks, Island, Columbia
3	SZA	Hip-Hop, Pop, R&B	TDE, RCA
4	Shakey Graves	Blues, Country, Folk, Rock	Dualtone
5	Bad Bunny	Hip-Hop, Latin Trap, Reggaeton	Hear This Music, Rimas

performances:

pconst	aconst	stage	start_time	duration
1	3	Main	17:00	60
2	1	Meadow	15:00	45
3	1	Main	21:00	90
4	5	Main	18:00	60
5	2	Meadow	17:00	60
6	4	Meadow	18:00	45
7	2	Main	20:00	60

(a) (2.0 pt)

Complete the query to retrieve the name and record labels of all artists who are in the Folk genre.

```
SELECT _____________________________ FROM artists WHERE _____________________________________ ;
```

(b) (3.0 pt)

Complete the query to display the name of each artist alongside the stage they performed on and the duration of the performance in order of longest to shortest (an artist can appear multiple times if they perform multiple times).

```
SELECT a.name, p.stage, p.duration

FROM ____________________________________________________________________________________

ON ______________________________________________________________________________________

ORDER BY ________________________________________________________________________________ ;
```

(c) (4.0 pt)

Complete the query to return the names of performers who are on stage for strictly more than 2 hours (i.e. more than 120 minutes) and also display the duration of their longest performance as column `longest_show`.

```
SELECT a.name, ____________________________________________________ AS longest_show

FROM artists AS a JOIN performances AS p ON ____________________________________________

GROUP BY ________________________________________________________________________________

HAVING __________________________________________________________________________________ ;
```

(d) (4.0 pt)

You write the following query:

```
SELECT first.start_time, first.stage AS stage1, second.stage AS stage2
FROM performances AS first JOIN performances AS second
ON first.pconst != second.pconst
WHERE first.start_time = second.start_time;
```

Complete the table this query returns. You may not need to use all the rows.

start_time	stage1	stage2

(e) (3.0 points)

Any single performance that is at least 60 minutes requires hiring stage hands at a different pay bracket.

Complete the query to find the total stage time (in minutes) per stage, but only include the time for performances that go for at least an hour. Label the total column `total_mins`, and sort by total minutes from highest to lowest.

```
SELECT stage, _______ FROM performances _______ ;
              (a)                       (b)
```

i. (1.0 pt) Fill in blank (a).

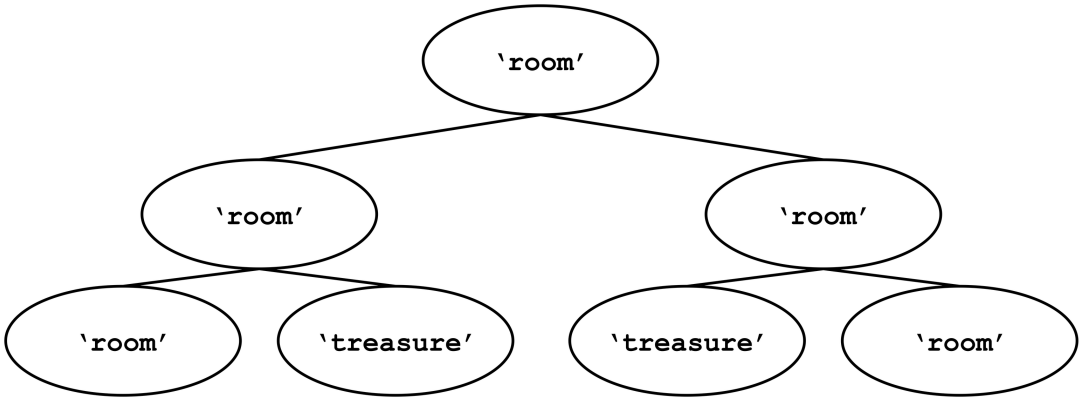
- ◯ `MAX(duration) AS total_mins`
- ◯ `SUM(duration) AS total_mins`
- ◯ `SUM(COUNT(*)) AS total_mins`
- ◯ `COUNT(*) AS total_mins`

ii. (2.0 pt) Fill in blank (b).

- ◯ `GROUP BY stage HAVING duration >= 60 ORDER BY total_mins DESC;`
- ◯ `GROUP BY pconst HAVING duration >= 60 ORDER BY total_mins DESC;`
- ◯ `GROUP BY stage HAVING duration >= 60 ORDER BY total_mins;`
- ◯ `WHERE duration >= 60 GROUP BY stage ORDER BY total_mins DESC;`
- ◯ `WHERE start_time - duration >= 60 GROUP BY stage ORDER BY total_mins;`

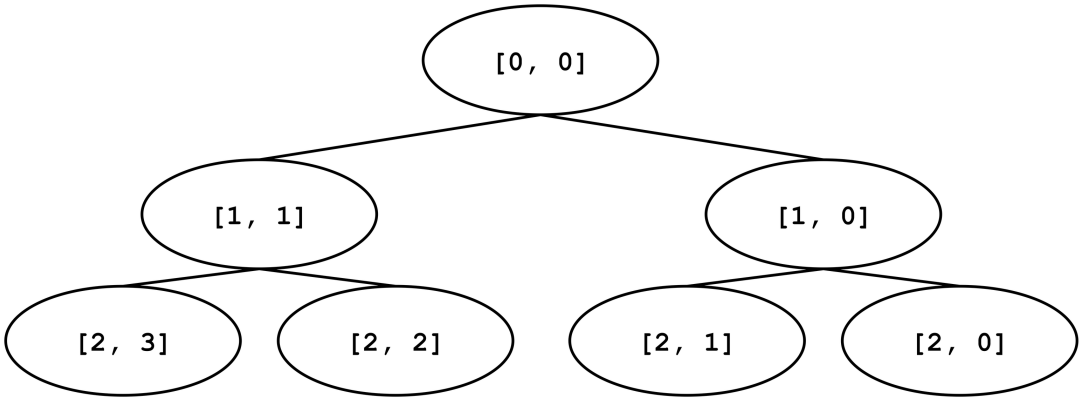
4. (9.0 points) Where's the Treasure?

The function `treasure_locations` accepts a single parameter `t`, an instance of the `Tree` class where all non-leaf nodes have two branches, the label of each node is either `'room'` or `'treasure'`, and only leaf nodes can have the `'treasure'` label. Here's a drawing of an example input tree:



The function returns a list of locations of treasures in `t`, where each location is a two-element list with the "depth" as the first element and the "right" as the second element. The "depth" of the root node is 0, and it increases by one at each level of the tree down from the root node. The right-most node of each level of the tree has a "right" of 0, and it increases from there (so it represents the number of nodes on that level which are to the right of the given node).

Here's the same tree where each node is labeled with its location:



When treasure_locations is called on the example tree, it returns `[[2, 2], [2, 1]]`, since the two treasures are located at `[2, 2]` and `[2, 1]`.

Complete the `treasure_locations` function below per the doctests and description.

Note: `sum([ [2, 3] ], [1]) = [1, 2, 3]`; the sum starts at `[1]`, then add `[2, 3] = [1, 2, 3]`.

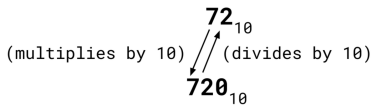
```
def treasure_locations(t):
    """
    Returns a list of location of treasures in the tree t, where each location
    is a two-element list, with the depth as the first element
    and the right as the second element. The depth of the root node is 0 and
    increases from there. The right starts at 0 from the right-most branch of each tree.
    Every non-leaf node has two branches.

    >>> t1 = Tree('room', [Tree('treasure'), Tree('room')])
    >>> print(t1)
    room
      treasure
      room
    >>> treasure_locations(t1)
    [[1, 1]]
    >>> t2 = Tree('room', [Tree('room', [Tree('room'), Tree('treasure')]),
    ...                    Tree('room', [Tree('treasure'), Tree('room')])])
    >>> print(t2)
    room
      room
        room
        treasure
      room
        treasure
        room
    >>> treasure_locations(t2)
    [[2, 2], [2, 1]]
    >>> no_treasures = Tree('room', [Tree('room'),
    ...                              Tree('room', [Tree('room'), Tree('room')])])
    >>> treasure_locations(no_treasures)
    []
    """
    def helper(t, depth, right):
        if t.label == 'treasure':

            return ____________________________________________________

        locations = [helper(

            ____________________________________________________________,

            ____________________________________________________________,

            ____________________________________________________________ + len(t.branches) - 1 - i

            ) for i in _____________________________________________]

        return sum(locations, [])   # Look at note at top of page

    ____________________________________________________________________
```

5. (6.0 points) I love this question to bits!

We want to write a procedure to convert a binary numeral (number in base 2) whose binary digits (bits) are stored in a list into decimal. Just as the value of a decimal numeral is simply each digit multiplied by the base (10) raised to increasing powers, the value of a binary numeral is each digit (bit) multiplied by the base (2) raised to increasing powers:

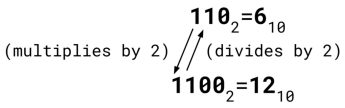
$$\begin{aligned}\mathbf{723}_{10} &= \mathbf{7} * 10^2 + \mathbf{2} * 10^1 + \mathbf{3} * 10^0 \\ &= \mathbf{7} * 100 + \mathbf{2} * 10 + \mathbf{3} * 1 \\ &= 700 + 20 + 3 \\ &= 723_{10}\end{aligned}$$

Fun fact: Just as shifting a decimal number to the *left* by one place multiplies it by 10 (and shifting to the *right* divides by 10):



$$\begin{aligned}\mathbf{1101}_2 &= \mathbf{1} * 2^3 + \mathbf{1} * 2^2 + \mathbf{0} * 2^1 + \mathbf{1} * 2^0 \\ &= \mathbf{1} * 8 + \mathbf{1} * 4 + \mathbf{0} * 2 + \mathbf{1} * 1 \\ &= 8 + 4 + 0 + 1 \\ &= 13_{10}\end{aligned}$$

…with binary numerals, shifting it to the *left* by one place multiplies it by 2 (and shifting it to the *right* divides it by 2):




```
scm> (binary-to-decimal '(1 1 0))
6
scm> (binary-to-decimal '(1 1 0 0))
12
scm> (binary-to-decimal '(1 1 0 1))
13
```

Fortunately, `reduce` is built into Scheme. Here's a review of how it works:

```
(reduce <combiner> <lst>)
```

Returns the result of sequentially combining each element in `lst` using `combiner` (a two-argument procedure). `reduce` works from left-to-right, with the existing combined value passed as the first argument and the new value as the second argument. `lst` must contain at least one item.

Examples:

```
scm> (reduce + '(2 4 1 3)) ;; this creates (+ (+ (+ 2 4) 1) 3)
10
scm> (reduce + '(2))       ;; reduce called on a list of one element returns that element
2
```

Fill in the blank to complete the procedure.

```
(define (binary-to-decimal bits)

    (reduce ____________________________________________________________________________ bits))
```

6. (10.0 points) I'm on a Quest to show what I know about sharks and pregnant fish!

This question can clobber Q3 on the Quest (i.e., a higher score here will replace a lower score there).

Complete the pure function `quest` to have the following behavior:

```
>>> four = quest(4)
>>> def fact(val, n): return val * n
>>> four(fact)       ## the product of 1 through 4 inclusive
24
>>> def plus(val, n): return val + n
>>> four(plus)       ## the sum of 1 through 4 inclusive
10
>>> def join(val, n): return val*10 + n
>>> four(join)       ## 1 through 4 concatenated together
1234
>>> def ones(val, n): return val*10 + val%10
>>> four(ones)       ## ignore n and duplicate the rightmost digit
1111
>>> def allv(val, n): return val
>>> four(allv)       ## ignore n and return val
1
>>> def alln(val, n): return n
>>> four(alln)       ## ignore val and return n
4
```

Please be aware of your indentations. We will be strict with syntax when grading this question. You may not need to use all the lines.

```
def quest(n):
```

---

7. (8.0 points) Scheme Trek III: The search for `(fib 100)`...

All the redundant computation of Fibonacci naive recursive implementation in Scheme becomes tedious, so you decide to modify the Scheme interpreter so it **remembers the return values of all function calls, and doesn't have to compute them twice.** For example, whereas the following would never return because it takes too long to run:

```
scm> (define (fib n) (if (< n 2) n (+ (fib (- n 1)) (fib (- n 2)))))
fib
scm> (fib 100)  ;; never returns
```

... after your change, we get:

```
scm> (fib 100)
354224848179261915075
```

Below are selected parts of the Scheme interpreter. You should modify the following code by selecting a line to replace (with perhaps more than one line). *Assume we'll never redefine functions and that all functions are pure.*

```
1    D = {}
2
3    def scheme_eval(expr, env):
4        ...
5        first, rest = expr.first, expr.rest
6        ...
7        procedure = scheme_eval(first, env)
8        args = rest.map(lambda operand: scheme_eval(operand, env))
9        return scheme_apply(procedure, args, env)
10
11    def scheme_apply(procedure, args, env):
12        ...
13        if isinstance(procedure, PrimitiveProcedure):
14            return apply_primitive(procedure, args, env)
15        elif isinstance(procedure, LambdaProcedure):
16            new_env = procedure.env.make_call_frame(procedure.formals, args)
17            return scheme_eval(procedure.body, new_env)
18        ...
```

(a) Which line do you want to replace? *Please write a line number e.g. 30*

(b) What code do you want to replace the removed line with? *You may write as many lines as needed.*

No more questions.