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?

1. **(8.0 points)   What Would Python Display?**

   (a) **(4.0 points)**

   Assume the following code has been executed.

   ```
   def os(ki):
       t = -1

   i = 5
   t = 7
   while i > 3:
       t = i - 4 and i + 4
       os(i - 2)
       i, j = i - 1, i * 2
       s = i
   ```

   i. (1.0 pt) What value is bound to `i` in the global frame?

   ii. (1.0 pt) What value is bound to `j` in the global frame?

   iii. (1.0 pt) What value is bound to `s` in the global frame?

   iv. (1.0 pt) What value is bound to `t` in the global frame?

(b) **(4.0 points)**

The function `tik` takes an argument `tok` and returns a function `insta` that takes an argument `gram`. The `insta` function prints `tok` and `gram` on the same line separated by a space and has no return statement. Its implementation has been omitted intentionally.

```
def tik(tok):
    """Returns a function that takes gram and prints tok and gram on a line.

    >>> tik(5)(6)
    5 6
    """
    def insta(gram):
        ... # The implementation of this function has been omitted.
    return insta
```

i. **(4.0 pt)** What would the interactive Python interpreter display upon evaluating the expression:

`tik(tik(5)(print(6)))(print(7))`

2. (5.0 points)   31 Cal Olympians

The environment diagram below was generated by code that is only partially provided to the right of the diagram. All of the relevant code needed to fill in the blanks in the environment diagram is shown. Line numbers have been omitted intentionally.

PFymZ5B.png

(a) **(1.0 pt)** Which of these could fill in blank (a)?

○ 3

○ 5

○ 7

○ 2021

(b) (1.0 pt) Which of these could fill in blank (b)? (The parent of the function returned in `f3`.)

○ `parent=Global`

○ `parent=f1`

○ `parent=f2`

○ `parent=f3`

(c) (1.0 pt) Which of these could fill in blank (c)?

○ `parent=Global`

○ `parent=f1`

○ `parent=f2`

○ `parent=f3`

(d) (1.0 pt) Which of these could fill in blank (d)?

○ 3

○ 5

○ 7

○ 2021

(e) (1.0 pt) Which of these could fill in blank (e)?

○ 3

○ 5

○ 7

○ 2021

3. (27.0 points)   All Hail the Stone

Definition: A *hailstone sequence* begins with a positive integer n. If n is even, divide it by 2. If n is odd, triple it and add 1. Repeat until 1 is reached. For example, the hailstone sequence starting at 3 is 3, 10, 5, 16, 8, 4, 2, 1. Assume all hailstone sequences are finite and end with 1.

Assume the following code has been executed.

```
from operator import add, mul


def next_hail(k):
    """Return the next element in a hailstone sequence."""
    assert k > 1
    if k % 2 == 0:
        return k // 2
    else:
        return 3 * k + 1
```

(a) (8.0 points)

Implement `hail_min`, which takes a positive integer n and a one-argument function `measure`. It returns the element of the hailstone sequence starting with n for which calling `measure` on the element returns the smallest value.

If more than one element of the sequence has the smallest `measure` value, return the earliest one.

```
def hail_min(n, measure):
    """Return the element k of the hailstone sequence starting with n for which
    measure(k) is smallest. In case of a tie, return the earliest element.

    >>> hail_min(5, lambda k: -k)         # Among 5, 16, 8, 4, 2, 1; 16 is largest
    16
    >>> hail_min(8, lambda k: -k)         # Among 8, 4, 2, 1; 8 is largest
    8
    >>> hail_min(3, lambda k: abs(k - 7))  # Among 3, 10, 5, 16, 8, 4, 2, 1; 8 is closest to 7
    8
    >>> hail_min(9, lambda k: abs(k - 7))  # Among 9, 28, 14, 7, 22, ...; 7 is closest to 7
    7
    >>> hail_min(8, lambda k: abs(k - 3))  # 4 and 2 are both close to 3, but 4 is earliest
    4
    """

    apple = _____
              (a)

    while n > 1:

        n = next_hail(n)

        if _____:
              (b)

            _____
              (c)

    return _____
             (d)
```

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

\
\
\
\

ii. (3.0 pt) Fill in blank (b).

\
\
\
\

iii. (1.0 pt) Fill in blank (c).

\
\
\
\

iv. (1.0 pt) Which of these could fill in blank (d)? Check all that apply.

☐ `n`

☐ `apple`

☐ `measure(n)`

☐ `measure(apple)`

☐ `min(n, apple)`

☐ `min(measure(n), measure(apple))`

v. (2.0 pt) What element of the hailstone sequence starting with **n** larger than 1 is returned by the expression:

`hail_min(n, lambda k: 1 - k % 2)`

○ Always **n** (the first element)

○ Always 1 (the last element)

○ The largest odd element

○ The largest even element

○ The smallest odd element

○ The smallest even element

○ The first odd element

○ The first even element

(b) (6.0 points)

Definition: An *accumulator function* is a function that takes two integers and returns an integer. It can serve as the second argument to `hail_tally` below.

```
def hail_tally(n, f):
    """Accumulate the elements of the hailstone sequence starting with n using
    accumulator function f.

    >>> hail_tally(3, add)    # 3 + 10 + 5 + 16 + 8 + 4 + 2 + 1 = 49
    49
    >>> hail_tally(10, max)   # Largest of 10, 5, 16, 8, 4, 2, 1
    16
    """
    total = 0
    while n > 1:
        total = f(total, n)
        n = next_hail(n)
    return f(total, 1)
```

Implement `sum_some`, which takes a one-argument function `select` and returns an *accumulator function* `f`. For positive integer `n`, the call `hail_tally(n, f)` returns the sum of the elements of the hailstone sequence starting with `n` for which calling `select` on the element returns a true value.

```
def sum_some(select):
    """Return an accumulator function that sums all k for which select(k) is a true value.

    >>> def below_ten(k):
    ...     return k < 10
    >>> sum_below_ten = sum_some(below_ten)
    >>> hail_tally(3, sum_below_ten)  # [3] + 10 + [5] + 16 + [8] + [4] + [2] + [1]
    23
    """
    def f(total, k):

        if _____:
                (a)

            return _____(total, k)
                       (b)

        return _____
                  (c)


    _____
       (d)
```

i. (2.0 pt) Fill in blank (a).

ii. **(1.0 pt)** Which of these could fill in blank (b)? Check all that apply.

☐ `add`

☐ `mul`

☐ `sum_some`

☐ `total`

☐ `f`

☐ `hail_tally`

iii. **(1.0 pt)** Which of these could fill in blank (c)?

○ `k`

○ `total`

○ `total + k`

○ `hail_tally(k, sum_some)`

○ `hail_tally(total, sum_some)`

○ `hail_tally(total + k, sum_some)`

iv. **(2.0 pt)** Fill in blank (d).

(c) **(5.0 points)**

Implement `hail_odd_sum`, a function that takes a positive integer `n` and returns the sum of all odd elements in the hailstone sequences starting with `n`.

Important: Your solution **must** include `sum_some`.

You may also call other functions defined previously in this question.

```
def hail_odd_sum(n):
    """Sum the odd elements of the hailstone sequence starting with n.

    >>> hail_odd_sum(3)    # [3], 10, [5], 16, 8, 4, [1]; 3 + 5 + 1 = 9
    9
    >>> hail_odd_sum(34)   # 34, [17], 52, 26, [13], 40, 20, 10, [5], ..., [1]
    36
    """
    return _____(_____, _____)
             (a)         (b)          (c)
```

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

ii. **(1.0 pt)** Fill in blank (b).

iii. **(3.0 pt)** Fill in blank (c).

(d) (8.0 points)

Definition: To call a function `f` *repeatedly* on a sequence of values `x`, `y`, `z` means to use `f` in a nested call expression in which each element of the sequence is passed in as a single argument in order: `f(x)(y)(z)`.

Implement `hail`, which takes an integer `n` greater than 1. It returns a function that returns `True` when called repeatedly on all of the remaining elements of the hailstone sequence starting with `n`. It returns `False` when called repeatedly on any sequence that differs from the hailstone sequence starting with `n`.

Hint: You may call `next_hail` (and other functions defined previously in this question).

```
def hail(n):
    """Return a function that returns True if called repeatedly on the remaining
    elements of the hailstone sequence starting with n and False otherwise.

    >>> hail(3)(10)(5)(16)(8)(4)(2)(1)
    True
    >>> hail(3)(4)                      # The next element should have been 10.
    False
    >>> hail(3)(10)(5)(16)(8)(1)    # The next element should have been 4.
    False
    """

    assert n > 1

    def check(k):

        if _____:
               (a)

            return True

        if _____:
               (b)

            return False

        return _____
                  (c)

    return _____
             (d)
```

i. (2.0 pt) Fill in blank (a).

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

iii. **(2.0 pt)** Fill in blank (c).

iv. **(2.0 pt)** Fill in blank (d).

No more questions.