CS131 Fall '24 Final

Dec 9th, 2024

Student ID #: _____-_____-_____

Full Name (First, Last): _____

Practice Academic Integrity - Don't cheat.
(There are multiple versions of the exam, so copying from a neighbor will only get you caught - trust us!)

Problem #1: Harder Haskell… Noooooooo! (15 min)	/13
Problem #2: Always Clean up After Yourself! (21 min)	/14
Problem #3: You're Bound To Do Well (12 min)	/12
Problem #4: How Sharp Are You On Brewin#? (15 min)	/14
Problem #5: Spooky Arithmetic (22 min)	/21
Problem #6: The Problem of Our Generation (25 min)	/23
Problem #7: Are you a Pro or a Log? (25 min)	/25
Total (110 min) (1 hr 50 min)	**/122**

Although we expect this exam to take around 110 min,
you will have 150 min (2.5 hrs)

1. Harder Haskell… Noooooooo! [13 Points]

Lambda calculus is the mathematical basis of functional programming. It contains formal theories related to lambda expressions. There is an interesting idea in lambda calculus called Church Encoding, which defines a sequence of lambda expressions with interesting properties. Such kinds of lambda expressions are called Church Numerals. Using Haskell as our tool, we can take a peek without going too far into the formal theories.

Part A

The first three (i.e., the "zeroth", first and second) Church Numerals in the sequence can be written in Haskell as:

```
c_0 = \f -> \x -> x
c_1 = \f -> \x -> f x
c_2 = \f -> \x -> f (f x)
```

Despite the name, the lambdas that define Church Numerals can be applied to more than just numeric types. You should not make any extra assumptions on the types of `f` and `x` in your answer.

A.1 Write down the type signature of `c_1`.

ANSWER: `c_1 :: _____`

It turns out that Church Numerals all have the same general form, the only difference is that the n-th Church Numeral, `c_n`, applies `f` on `x` for `n` times (`n` = 0, 1, 2, …).

A.2 Write a one-line function named *`church_to_index`* that returns the (zero-based) index given an arbitrary Church Numeral `c`. For example:

```
church_to_index c_0        -- returns 0 when using c_0 from above
church_to_index c_2        -- returns 2 when using c_2 from above
```

Hint: Remember that c is a function that accepts another function as its argument and returns a function

ANSWER: `church_to_index c = _____`

Part B

We can also write a function to obtain an arbitrary Church Numeral given its index:

```
get_nth_church n = church_gen n c_0
  where
    c_0 = \f -> \x -> x
    succ c f = \x -> f (c f x)
    church_gen 0 cacc = cacc
    church_gen x cacc = church_gen (x - 1) (succ cacc)
```

The key here is the helper function *succ* (successor). It takes a Church Numeral and returns its successor (i.e., the next Church Numeral in sequence). That is to say, `(succ c_0)` evaluates to `c_1`, and `(succ c_1)` evaluates to `c_2`, etc. The whole function works by applying `succ` on `c_0` for `n` times recursively to obtain `c_n`.

What are the free variable(s) in the lambda expression *`\x -> f (c f x)`* as returned by the *succ* function? (If there are no free variables, answer None).

ANSWER: _____

Part C

C.1 Ashley wants to write a function *first_n_church* that returns the first `n` Church Numerals (starting from `c_0`) in a list. Please help her write an one-line implementation using a **list comprehension** and the *get_nth_church* function above:

ANSWER:

```
first_n_church n = [ ____________________________________________________ ]
```

C.2 Ashley thinks the efficiency of the function above can be improved from $O(n^2)$ to $O(n)$. **Please fill in the blanks in the code below to satisfy her big-O requirement.** (We assume `c_0` and `succ` have already been defined as in Part B)

```
first_n_church n = ________________ (helper 0 [])
  where
    helper x accum
      | x >= n    = ________________
      | x == 0    = helper ________________ [c_0]
      | otherwise = helper (x + 1)
                           ((succ _________________) : _________)
```

2. Always Clean up After Yourself! [14 Points]

Suppose we run this code that attempts to create a linked list, written in a mystery language that uses **object reference semantics**:

```
[01] require "addr" # gets the memory address of an object
[02]
[03] class Node
[04]   fields val, prev, next
[05]   def initialize(val)
[06]     @val = val   # Similar to self.val = val in Python
[07]     @prev = nil
[08]     @next = nil
[09]   end
[10] end
[11]
[12] class LinkedList
[13]   fields head
[14]   def initialize
[15]     @head = nil
[16]   end
[17]
[18]   def populate
[19]     @head = Node.new(67)
[20]     n2 = Node.new(68)
[21]     n3 = Node.new(69)
[22]     n4 = Node.new(70)
[23]     n5 = Node.new(71)
[24]     # print "#{addr(n2)}"
[25]
[26]     @head.next = clone(n2) # shallow copy
[27]     n2.next = n3
[28]     n3.next = n4
[29]     n4.next = n5
[30]     n5.prev = n3  # prev, not next!
[31]     # print "#{addr(n2)}"
[32]   end
[33] end
[34]
[35] ll = LinkedList.new
[36] ll.populate
[37] ll = nil
```

Part A. For part A, you may assume that this language uses **reference counting** to do garbage collection and that garbage collection happens immediately after an object is no longer referenced.

A.1 Which nodes, if any, are still active and in memory right **before** `populate` returns? Write the value (e.g., 68) of each node that's still active or None if there are none. If multiple active nodes have the same value, you must write the value multiple times.

ANSWER: _____

A.2 Which nodes, if any, are still active and in memory right **after** `populate` returns? Write the value (e.g., 68) of each node that's still active or None if there are none. If multiple active nodes have the same value, you must write the value multiple times.

ANSWER: _____

A.3 Which nodes, if any, are still active and in memory right **after the last line of the program** executes but before the program terminates? Write the value (e.g., 68) of each node that's still active or None if there are none. If multiple active nodes have the same value, you must write the value multiple times.

ANSWER: _____

Part B. Now assume ***hypothetically*** that this language uses **mark and compact** to do garbage collection:

B.1 Which Node objects, if any, are ***reachable*** right **before** `populate` returns? Write the value of each of them or None if there are none. If multiple reachable nodes have the same value, you must write the value multiple times.

ANSWER: _____

B.2 Which Node objects, if any, are ***reachable*** right **after** `populate` returns? Write the value of each of them or None if there are none. If multiple reachable nodes have the same value, you must write the value multiple times.

ANSWER: _____

B.3 Which Node objects, if any, are ***reachable*** right **after the last line of the program** executes but before the program terminates? Write the value of each of them or None if there are none. If multiple reachable nodes have the same value, you must write the value multiple times.

ANSWER: _____

Part C.

Now assume that we do not know the *actual* garbage collection strategy this language uses, but we know that it is one of the approaches we learned in class. It's your job to figure out which one by observing the behavior of variable **n2** throughout the program's execution.

Suppose that when we uncomment lines 24 and 31 and run the program, the program outputs:

720
360

What garbage collection strategy does this language actually use? In one sentence, explain how you determined this.

ANSWER: _____

_____

3. You're Bound To Do Well [12 Points]

You've been given a program in an unusual language which supports value, reference *and* object reference binding semantics. As you can see, the language explicitly specifies the **binding semantics** for every variable and parameter:

```
func foo(ref a, val b) {
  a = Person(name: "Jaquin");
  b.name = "Monte";
}

func bar(objref x, ref y) {
  x.name = x.name + "poo";
  foo(x, y);
  print(x.name, y.name);

  x = y;
  y = Person(name: "Lu");
}

func bletch(val p, objref q) {
  q.name += "ala";
  bar(p, q);

  print(p.name, q.name);
  p.name += "ito";
}

func main() {
  objref m = Person(name: "Samantha");
  objref n = Person(name: "Brent");
  bletch(m, n);
  print(m.name, n.name);
}
```

Notes:

- If z is a reference to an object reference x, changes to z modify the object reference and not the object pointed to by the object reference.
- When passing an object reference to a function that accepts a value parameter, the language passes a copy of the object pointed to by the object reference
- When passing a value to a function that accepts an object reference, an object reference to that value is passed

What does the above program print?

ANSWER:

_____

_____

_____

4. How Sharp Are You On Brewin#? [14 Points]

Consider this Brewin# program which was written to test project 4:

```
func foo(a) {
  print("voldemort");
  if (1/a == 1) {
    print("valerian");
    return 5;
  }
  print("cedric");
  return 10;
}

func bar(a) {
  print("jordan");
  a = 20;
  return a;
}

func main() {
    var x;
    var y;
    var in;
    in = inputi();

    if (in == 1) {
        x = 1;
        y = bar(x) + x;
    }
    if (in == 2) {
        x = foo(1) - 5;
        y = foo(2) + x + x;
    }
    if (in == 3) {
        x = bar(1);
        y = foo(x);
        x = y;
    }

    try {
        foo(x);
        print(y);
    }
    catch "div0" {
        print("severus");
    }
}
```

For each part/subproblem below, you may assume that:

- the program is **executed from scratch** in a fully working **project 4** interpreter, independently of the other subproblems
- the subproblem specifies what input is provided to the program (e.g., 1, 2 or 3) during its execution

For each subproblem below, you must write the output of the program assuming the input is the value specified. If you believe the program crashes at any point (due to an uncaught exception, or any other error that violates the rules of Brewin#), **write all outputs up until the point that you think the program crashes and then write "crash" as your final line**.

For each part, write each output on a separate line.

Part A. Suppose the user inputs a value of 1, what is the program's output?

ANSWER:

Part B. Suppose the user inputs a value of 2, what is the program's output?

ANSWER:

Part C. Suppose the user inputs a value of 3, what is the program's output?

ANSWER:

5. Spooky Arithmetic [21 Points]

Consider this code snippet from a ***compiled***, made-up language:

```
[01] type BasicArith {
[02]   Add (+) (op2 BasicArith) BasicArith
[03]   Subtract (-) (op2 BasicArith) BasicArith
[04] }
[05]
[06] func Sum[T any](values []T) T {
[07]   var total T
[08]   for _, v := range values {
[09]     total = total (+) v
[10]   }
[11]   return total
[12] }
[13]
[14] func main() {
[15]   arr := []SpookyInt{10, 20, 30} // SpookyInt definition not shown
[16]   fmt.Println(Sum(arr))
[17]   // fmt.Println(arr[2] (-) arr[1])
[18] }
```

The compiler for this language outputs the following error during compilation:

```
operator (+) not defined on a variable of type any
```

However, when **any** on line 6 is changed to **BasicArith**, the code compiles, and when executed, outputs 60 as expected.

Part A

For all of Part A, you may assume that we changed the term "any" on line 6 to "BasicArith" and that the code outputs 60 as expected.

A.1 Without making any additional assumptions (such as compiler optimizations), identify all of the following programming language strategies that the code must be using to operate as described and write them in the answer section. You will lose points for additional incorrect selections:

A. Parametric polymorphism
B. Subclass inheritance
C. Implementation inheritance
D. Dynamic dispatch
E. Duck typing
F. Templates
G. Generics

ANSWER: _____

A.2 Suppose we uncomment line 17. Will the program compile (yes, no, can't be determined)? Explain your answer using a **maximum of 2 sentences**. Explanations longer than 2 sentences will not receive credit.

ANSWER:

_____

_____

_____

A.3 Suppose we change our main function to look like this - notice it uses a new type called *FriendlyType* which is defined for use, but its definition is not shown:

```
func main() {
  arr := []FriendlyInt{10, 20, 30} // FriendlyInt definition not shown
  fmt.Println(arr[2] (-) arr[1])
  // fmt.Println(Sum(arr))
}
```

This code compiles and outputs 10 when run.

Now suppose we uncomment the commented line. Will the program compile (yes, no, can't be determined)? Explain your answer using a **maximum of 2 sentences**. Explanations longer than 2 sentences will not receive credit.

ANSWER:

_____

_____

_____

Part B Consider this snippet of Python code:

```
[01] class BasicArith:
[02]     def __init__(self, v):
[03]       self.v = v
[04]     def __add__(self, op2):
[05]       return BasicArith(self.v + op2.v)
[06]     def __sub__(self, op2):
[07]       return BasicArith(self.v - op2.v)
[08]     def __str__(self):
[09]       return str(self.v)
[10]
[11] class SpookyInt(BasicArith):
[12]     def __init__(self, v):
[13]       super().__init__(v)
[14]
[15] def Sum(values) -> BasicArith: # should return BasicArith object
[16]     total = 0
[17]     for v in values:
[18]         total = total + v
[19]     return total
[20]
[21] def main():
[22]   arr = [SpookyInt(n) for n in [10,20,30]]
[23]   print(Sum(arr))
[24]   # print(arr[2] - arr[1])
[25]
[26] if __name__ == '__main__':
[27]   main()
```

Hints: In Python, calling a + b actually calls a.__add__(b) under the hood and calling a - b actually calls a.__sub__(b) under the hood

B.1 This code crashes when it runs. Modify a **single line** of code such that the program will output “60” as expected and the implementation of **Sum** follows the type hint given on line 15.

ANSWERS:

What line # do you chose to modify: _____

Show the corrected version of the line: _____

For parts B.2, B.3 and B.4 of the problem, assume your answer to B.1 is correct.

B.2 Identify all of the following programming language strategies that are used in the example above and write them in the answer section. You will lose points for additional incorrect selections:

A. Parametric polymorphism
B. Subclass inheritance
C. Implementation inheritance
D. Dynamic dispatch
E. Duck typing
F. Templates
G. Generics

ANSWER: _____

B.3 Suppose we uncomment line 24. Will the program crash when run (**yes**, **no**, **can't be determined**)? **Explain your answer** using a **maximum of 2 sentences**. Explanations longer than 2 sentences will not receive credit.

ANSWER:

_____

_____

_____

B.4 Suppose we change our main function to look like this:

```
def main():
    arr1 = [SpookyInt(n) for n in [10,20,30]]
    # FriendlyInt definition not shown, but exists in our source file
    arr2 = [FriendlyInt(n) for n in [10,20,30]]
    print(arr1[0] + arr2[0])
    # print(arr2[0] + arr2[1])
```

The code outputs 20 when run.

Now suppose we uncomment the bottom commented line. Will the program crash when run (**yes**, **no**, **can't be determined**)? **Explain your answer** using a **maximum of 2 sentences**. Explanations longer than 2 sentences will not receive credit.

ANSWER:

_____

_____

_____

6. The Problem of Our Generation [24 Points]

Note: **all parts of this problem refer to Python.**

Part A

Fill in the blanks to create a generator for the Fibonacci sequence 1, 1, 2, 3, 5, 8, … (i.e., the first two elements are ones, and each next element is the sum of the two preceding elements). Given a value of n, your generator must generate all the elements *less than* n in increasing order and use constant memory.

Your generator should support the following syntax:

```
for i in fib(13):
    print(i, end=" ")
# prints 1 1 2 3 5 8 since each of these values is less than 13
```

ANSWER:

```
def fib(n):
    fib_prev = 1
    fib_cur = __________
    while ____________________:
        ____________________
        old_fib_prev = fib_prev
        fib_prev = __________
        fib_cur = __________
```

Part B

What does the following code output?

```
def take(g, n):
    lst = []
    lst.append(next(g))
    for i in range(n):
        lst.append(next(g))
    return lst

def gen1(a, d):
    while True:
        yield a
        a += d

print(take(gen1(300, 6), 4))
```

ANSWER: _____

Part C

Below is a mostly-completed definition of a *Matrix* class that represents a matrix as a list of lists. You may assume that the provided list will always be of the correct format (each row is of the same length, all lists only contain integers).

```
class Matrix():
    def __init__(self, vals):
        self.matrix = vals

    def get(self, row, col):
        return self.matrix[row][col]

    def set(self, row, col, val):
        self.matrix[row][col] = val

  ...
```

For problems C.1 and C.2, your goal is to be able to use this Matrix class as follows:

```
mat = Matrix([[0,1,2], [3, 4, 5]])
s = 0

for val in mat:
    s += val

print(s) # prints 15
```

C.1 Complete the definition of *MatrixIterator*, which is an iterator class for your new Matrix class, so that the matrix is traversed in a row-major order (i.e., first row left to right, then second row left to right, etc).

```
class MatrixIterator():
    def __init__(_____________________):
        self.matrix = _______________
        self.n_rows = len(vals)
        self.n_cols = _______________ if _______________ else 0
        self.row = 0
        self.col = 0

    def __________(self):
        if self.row == self.n_rows:
            ___________________________________
        val = ____________________
        self.col += 1
        if _________________:
            self.row = __________
            self.col = __________
        return val
```

C.2 Next, fill in the blanks below to complete the definition of the *Matrix* class so it supports iteration over its elements using *MatrixIterator*:

```
class Matrix():
    def __init__(self, vals):
        self.matrix = vals

    ...  # rest of code shown above

    def _________________(self):
        return ____________________________________________
```

Part D

Professor N was trying to implement a generator function `interleaved` that takes two iterable objects `iterable1` and `iterable2` and generates elements by alternating between them: first element of `iterable1`, first element of `iterable2`, second element of `iterable1`, etc. Elements of each iterable object should appear in the generated sequence in the order of iteration. Finally, when one of the iterables is exhausted, the generator should yield the remaining elements of the other iterable. This is how the function is intended to work.

```
lst1 = [1, 3]
lst2 = ["a", "b", "c", "d"]

for val in interleaved(lst1, lst2):    # line A
    print(val, end=" ")

# supposed to print 1 a 3 b c d
```

When Professor N asked a questionable LLM to give a solution, here's what it suggested:

```
def interleaved(iterable1, iterable2):
    for val1 in iterable1:
        yield val1
        for val2 in iterable2:
            yield val2
```

What is the result of running the loop above (labeled as **line A**) with this implementation of `interleaved`?

ANSWER: _____

7. Are you a Pro or a Log? [25 Points]

Prolog, especially its list processing, was a big headache for Johnny when he took CS131. But he managed to overcome it by using functional programming for inspiration. When he wants to write a list processing predicate in Prolog, he first tries to write an “equivalent” function in Haskell using pattern matching. It turns out that the latter can often be mapped into Prolog logic quite naturally.

For example, Jonny first wrote a list reverse function in Haskell on the lower left. Then, on the lower right he showed his functioning `do_rev` predicate in Prolog. We can see the close resemblance of the two versions.

Haskell	Prolog

`do_rev [] = []`
`do_rev (x:xs) =`
`  let t = do_rev xs`
`  y = t ++ [x]`
`  in y`
	`do_rev([], []).`
	`do_rev([X|Xs], Y) :-`
	`  do_rev(Xs, T),`
	`  append(T, [X], Y).`

Part A

Assume that we define `do_rev` as above and execute the query*:*

```
do_rev([1, 2, 3], [3, 2, 4])
```

in the Prolog interpreter.

A.1 What are the first two subgoals that Prolog will add to its evaluation stack in order to evaluate the `do_rev` rule?

ANSWER:

_____

_____

A.2 What are the mappings in Prolog’s evaluation stack when it has the above subgoals as *pending* goals (for example, Q -> [apple, pear]):

ANSWER: _____

Part B

Johnny was a little concerned about the efficiency of his Prolog predicate because he knows the Haskell version has $O(n^2)$ complexity due to the concatenate (++) operation. Because of this, he decided to write a new predicate named `do_rev2` that's guaranteed to have O(n) complexity, based on this more efficient Haskell version of reverse:

```
rev_app [] acc = acc
rev_app (q:qs) acc = rev_app qs (q:acc)

do_rev2 q = rev_app q []
```

Please help Johnny write a working `do_rev2 predicate` by applying his method. Fill in the blanks below:

ANSWER:

```
rev_app(_____________________, Acc, _____________________).
rev_app([Q|Qs], Acc, R) :-
         rev_app(_________________, _________________, R).

do_rev2(Q, R) :- rev_app(Q, _____________, _____________).
```

Part C

Now Johnny is getting excited and wants to write his own Prolog function without starting with a Haskell version. He's decided to implement a predicate to interleave two Prolog lists together. When interleaving two lists L1 and L2, the `interleave` predicate alternates elements from each list one by one: L1[0], L2[0], L1[1], L2[1], L1[2], L2[2], ... If one list runs out of elements before the other, the remaining elements of the longer list are automatically appended to the end of the resulting interleaved list.

For example:

?- interleave([a, b, c], [1, 2, 3], Result).
Result = [a, 1, b, 2, c, 3].

?- interleave([x, y, z, w], [7, 8], Result).
Result = [x, 7, y, 8, z, w].

?- interleave([], [alpha, beta, gamma], Result).
Result = [alpha, beta, gamma].

Complete the following Prolog code by filling in the blanks to define a predicate `interleave(List1, List2, Result)` that interleaves elements from `List1` and `List2` into `Result`. If one list is longer, the remaining elements are appended at the end.

ANSWER:

```
interleave(_________________, ListA, ListA).
interleave(ListB, _________________, ListB).
interleave([M|N], [J|K],
      [_________________, _____________ | _____________]) :-
      interleave(_________________, _________________, Result).
```

END OF EXAM