CS131 Spring 23 - Final Exam

June 15th, 2023

Full Name (**LAST**, FIRST): _____

Student ID: _____

Signature: _____

Problem #1: Haskell	/12
Problem #2: Haskell	/16
Problem #3: Memory management	/14
Problem #4: Parametric poly/duck typing	/10
Problem #5: Prolog	/8
Problem #6: Binding semantics	/15
Problem #7: Error Handling	/8
Problem #8: OOP	/6
Problem #9: This n that	/9
Problem #10: Data musings	/8
Problem #11: OOP Typing	/ 9
Total	**/115**

Please practice academic integrity. Don't cheat.

1. Haskell – chewing on choose (12 points)

The binomial coefficient (or “choose” function) is an important tool in math and computer science. As a refresher, the binomial coefficient can be written as a function of two variables n and k. Mathematically, we’d say

```
binomial(n, k) = n! / [k!(n-k)!]
```

Throughout this problem, we’ll build up a neat set of tools to work with binomials!

For all subproblems where you write a function, you must include a type annotation. Unless otherwise stated, use `Integer` when representing a numeric type.

Part A (3 points)

a. Write **two** functions:

- A function called ***fact*** that takes in one integer and returns its factorial. For numbers that are less than 0, return the number 0; you may (or may not) recall that 0! = 1.
- A function called ***binom*** that takes in two numbers, n and k, and returns the binomial coefficient as defined above – using ***fact***.

You may assume that n >= k >= 0 (i.e., your solution does not have to work on negative n or k). For example:

```
fact 0 = 1          fact 4 = 24
binom 1 1 = 1       binom 6 4 = 15
```

Write your **fact** and **binom** functions here:

Part B (3 points)

Write a function called ***binomials*** that takes in one number, n, and returns a list of each binomial coefficient for k = [0, n], inclusive, in order. You must use your function from Part A. For example:

```
binomials 1 = [1, 1]
binomials 4 = [1, 4, 6, 4, 1]
```

You may assume that n >= 0 (i.e., your solution does not have to work on negative n). Your solution must be no more than 3 lines long (including the type annotation).

Write your solution here:

Part C (2 points)

"Pascal's triangle" is a beautiful (infinite) mathematical object that involves binomials. We can represent Pascal's triangle as a list of lists, where the i^{th} list is the result of calling binomials i (starting from i = 0).

Fill in the blank below for the following infinite list comprehension, which should generate Pascal's triangle. You must use the *binomials* function you defined in Part B.

Here's how your comprehension might be used:

```
ghci> take 5 pascalsTriangle
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

```
pascalsTriangle :: [[Integer]]
pascalsTriangle = [ _____________________________________ ]
```

Part D (4 points)

Finally, write a function ***hasDivisible*** that takes in one argument n, and returns *only* the rows of Pascal's Triangle that have at least one element evenly divisible by that number. For example:

```
ghci> take 2 (hasDivisible 3)
[[1, 3, 3, 1], [1, 4, 6, 4, 1]]

ghci> take 2 (hasDivisible 2)
[[1, 2, 1], [1, 4, 6, 4, 1]]
```

You may assume that n >= 0 (i.e., your solution does not have to work on negative n). You may use a helper function if needed.

Write your solution here:

2. Haskell – parsing parsers (16 points)

For your class projects, we provided you a parser. In this problem, we'll implement some parser-related functionality – and see how functional programming makes writing parsers simple! **For all functions you write, you must include a type annotation.**

Part A (8 points)

In almost all languages, keeping track of parentheses is an important task. Write a predicate function ***hasBalancedParens*** that takes in a list of strings (i.e., tokens in a program), and returns true if and only if the parentheses in the string are "balanced". There are 2 requirements:

1. The number of opening and closing parentheses should be the same (including 0).
2. Every closing parenthesis must have exactly one matching open parenthesis to its left.

```
hasBalancedParens ["hello world"] = true
hasBalancedParens ["print", "(", "3", "+", "5", ")"] = true
hasBalancedParens ["print", "(", "3", "+", "5"] = false
hasBalancedParens ["print", "(", "3", "+", "5", ")", ")"] = false
hasBalancedParens ["print", ")", "3", "+", "5", "("] = false
```

You may use helper function(s). Write your solution here:

Part B (8 points)

Expression evaluation in programming languages is often represented as a tree. Sometimes, you can apply optimizations to these trees to make computation faster. One such optimization is zero-product replacement, where you replace any term multiplied by the constant 0 with 0 itself.

Consider this ADT, which only supports addition and multiplication:

```
data Expr = Multiply Expr Expr | Add Expr Expr | N Integer
```

We could represent the following prefix expression:

(* (+ 3 5) (+ 0 0)) as: `(Multiply (Add (N 3) (N 5)) (Add (N 0) (N 0)))`

Write a function called ***zeroOptimizer*** that takes in an *Expr* and returns an equivalent *Expr*, but with zero-product replacement. That is, whenever one or both of a Multiply expression's children starts out as a number with a value of zero, or has been converted by a recursive call to zeroOptimizer into a number with a value of zero, replace the Multiply variant with `(N 0)`, e.g.:

```
ghci> zeroOptimizer (Multiply (N 5) (Add (N 0) (N 0)))
(Multiply (N 5) (Add (N 0) (N 0)))
ghci> zeroOptimizer (Add (Multiply (Add (N 1) (N 2)) (N 0)) (N 6))
(Add (N 0) (N 6))
ghci> zeroOptimizer (Multiply (Add (N 3) (N 5)) (Multiply (N 0) (N 0)))
(N 0)
```

You may write a helper function. Write your solution here:

3. Memory Management (14 points)

All of these subproblems are about memory management in real life. The questions are moderately open-ended, so any reasonable and well-justified answer will receive full marks.

Part A (4 points)

Apple designed the Swift programming language to, in part, make it easier to develop user-facing applications like iOS and macOS apps. When deciding on Swift's memory management approach, the language designers picked "Automatic Reference Counting" (ARC): an implementation of the reference counting algorithm we discussed in class.

For the device user's experience – particularly in runtime performance – why might Swift's designers pick reference counting over a tracing algorithm (e.g. mark-sweep, mark-compact)? - Your answer must reference both approaches, and you must limit your answer to five sentences.

Write your answer here:

Part B (6 points)

In many practical applications, the amount of memory used by a program is the most important constraint to optimize for (as opposed to performance). For example, many embedded systems – like the microcontroller on an air conditioning unit – have very little onboard memory.

Let's assume we're writing a program and can choose the memory management scheme we use. Consider the four types of memory management that we have discussed in class:

- Manual memory management (like in C++)
- Reference Counting
- Mark and Sweep
- Mark and Compact

Rank each memory management approach from best to worst in terms of memory overhead. In particular, assume that your program is correctly implemented (i.e. it has no bugs); justify your ranking with a concise argument touching on either average-case or worst-case memory usage.

To receive full credit, you must discuss all four approaches. Write your answer here:

Part C (4 points)

Consider a language which uses mark and sweep garbage collection, has **mutable** variables, and supports concurrency. What issues might the garbage collection system run into when running programs that use concurrency, and how might we address these issues?

Limit your answer to five sentences. Write your answer here:

4 Parametric Polymorphism and Duck Typing (10 points)

Part A (6 points)

Between C++ templates and Java-like generics, is one approach “more conservative” than the other? In other words, does either approach disallow code at compile-time that could technically be valid code? Justify your answer with an example (it does not need to be actual code).

Part B (4 points)

Now consider a **bounded** generic function in a language like Java and a function that uses duck typing in a language like Python, both of which accept an object reference as a parameter. Which one is more likely to result in **runtime errors** due to an invalid method being called through the object reference? Justify your answer.

5. Prolog (8 points)

Consider the following Prolog program that defines a knowledge base of courses, their prerequisites, and courses that are required by the major:

```
prerequisite(cs31, cs32).
prerequisite(cs32, cs33).
prerequisite(cs33, cs35l).
prerequisite(cs33, cs131).
prerequisite(cs33, cs151b).

major_requirement(computer_science, cs131).
major_requirement(electrical_engineering, cs151b).
```

Write a Prolog predicate called ***is_major_prereq(Course, Major)*** that determines if a given course is *directly or indirectly* a prerequisite course for a course required by a major. You must ensure correct syntax for full points.

```
Example query:  is_major_prereq(cs31, computer_science)
Expected output: true     % cs31 -> cs32 -> cs33 -> cs131

Example query:  is_major_prereq(cs35l, electrical_engineering)
Expected output: false   % cs35l isn't a prereq for any EE major reqts.
```

X

X

X

Write your solution on the next page:

X

X

X

Solution for Problem #5:

6. Binding Semantics (15 points)

Consider this program in a hypothetical language called !Brewin:

```
struct Dog:
  String name
  int bark

def foo(Dog d) -> void:
  d.bark = 20
  d = Dog{"Koda", 5}
  d.bark = 10

def bar() -> int:
  print("bar")
  return 1 + 2

def main() -> void:
  var d1 = Dog{"Kippy", bar()}
  var d2 = d1
  foo(d1)
  // print results
  print("d1.name = ", d1.name, "d1.bark = ", d1.bark)
  print("d2.name = ", d2.name, "d2.bark = ", d2.bark)
```

Part A (3 points)

What would this program print if the language were using **object reference semantics** for all variable bindings (i.e., d1 and d2) and **pass-by-object reference** for all parameter passing?

Part B (3 points)

What would this program print if the language were using **value semantics** for all variable bindings (i.e., d1 and d2) and **pass-by-reference** for all parameter passing?

Part C (3 points)

What would this program print if the language were using **value semantics** for all variable bindings (i.e., d1 and d2) and **pass-by-value** for all parameter passing?

Part D (3 points)

What would this program print if the language were using **object reference semantics** for all variable bindings (i.e., d1 and d2) and **pass-by-reference** for all parameters
(e.g., in the call foo(d1), d would be a reference to the object reference d1)?

Part E (3 points)

At the moment the program reaches the *"// print results"* line (before the last 2 print statements execute), what would this program have printed if the language were **lazy** like Haskell and used **name semantics** for all variable definitions/assignments and **pass-by-name** for all parameters?

7. Error Handling (8 points)

Consider the following Python program:

```
def foo(x, y):
  print('Alpha')
  print(x / y)
  print('Beta')

def bar(x, y):
  print('Gamma')
  foo(x, y)
  print("Delta")

def main():
  bar(10, "foo")
  bar(10, 0) # Line Q
```

Part A (2 points)

Prior to any exception that terminates the program, what does this program print?

Part B (6 points)

Assume that Python has only three types of exceptions which can be caught: *ZeroDivisionError*, *TypeError*, and *Exception*. Make the smallest/simplest possible change to the above program by adding exception handler(s) such that it prints out the following **lines** *before* it terminates due to an exception during the call on line Q:

```
Gamma
Alpha
Phew!
Delta
Gamma
Alpha
<program terminates due to an exception>
```

You MUST NOT make any other changes beyond adding exception handler(s) which somehow print "Phew!". You **MUST NOT** use any ***finally*** **clauses** or **return** statements in your handler.

Write the full, updated function(s) here, showing only the function(s) that you changed:

8. Object Oriented Programming (6 points)

Consider the following C++ program:

```
class A {
public:
  virtual void foo() { cout << "A"; }
  void bar() { foo(); c++; }
  void bletch() { cout << c; }
private:
  static int c;  // class variable
};

class B : public A {
public:
  virtual void foo() { cout << "B"; }
};

int A::c = 0;  // c starts out at zero

int main() {
  A* v[2] = { new A(), new B() };

  for (int i = 0; i < 2; i++) {
    v[i]->bletch();
    v[i]->foo();
    v[i]->bar();
  }

  for (int i = 0; i < 2; i++)
    v[i]->bletch();
}
```

What is the output of this program?

9. This n That (9 points)

Part A (3 points)

Consider the following code in C++:

```
int a() { cout << "A"; return 4; }
int b() { cout << "B"; return 3; }
int c() { cout << "C"; return 2; }
int d() { cout << "D"; return 1; }
int e() { cout << "E"; return 5; }

int main() {
  if (a() > 10 || (b() > 2 || c() > 7) || d() < 7 && e() > 0)
    cout << "F";
}
```

What does it print (choose one)?

A. ABDEF
B. ADF
C. ABCD
D. ABF
E. ADEF

ANSWER: _____

Part B (3 points)

Given the *foo* function in Haskell below:

```
foo _ []     = []
foo f (x:xs) = f x : foo f xs
```

Which of the following is a valid type signature of the foo function (choose one)?

A. foo :: (a -> b) -> [a] -> b
B. foo :: (a -> b) -> ([a] -> [b])
C. foo :: (a -> b) -> ([a] -> b)
D. foo :: ([a] -> [b]) -> ([a] -> [b])
E. foo :: ((a -> b) -> [a]) -> [b]

ANSWER: _____

Part C (3 points)

What does this Python code print out?

```
def foo(x):
 print("B")
 for i in x:
   yield i
   print("C")

a = foo([1,2,3])
print("A")
print(next(a))
print(next(a))
```

What does it print (circle one)?

A. B, A, 1, C, 2
B. B, A, 1, C, 2, C, 3
C. A, B, 1, C, 2
D. A, B, 1, C, 2, C
E. B, A, 1, C, 2, C

ANSWER: _____

10. Data musings (8 points)

Consider the following code from a hypothetical language, which compiles without errors. When you run it, it executes without any errors up until it throws an exception on line Q:

```
// Animal and Dog class definitions not shown for brevity
func print_ages(Animal arr[]): void {
  for (for i = 0; i < NUM_ANIMALS; ++i)
    print(arr[i].age)
}

func main(): void {
  var NUM_ANIMALS = 10
  Animal pack[NUM_ANIMALS]    // An array of Animal object references
  pack[0] = new Dog(name: "Koda", age: 1)
  pack[1] = new Dog(name: "Kippy", age: 7)
  print_ages(arr: pack)
  pack[0].age = -10          // Line Q
}
```

Part A (2 points)

What can we conclude about the **scoping approach** used by this language? Why?

Part B (2 points)

Assuming that this language only creates new objects when the *new* command is explicitly used (e.g., *`new Dog(name: "Koda", age: 1)`*), what can we conclude about all of the **typing systems** used by this language (e.g., static, dynamic, …)? Why?

Part C (2 points)

What can we conclude about the Dog class given the program's behavior (as described above)?

Part D (2 points)

Does this program perform any **conversions or casts**? If so, list those that you find, detailing whether they are widening or narrowing.

11. OOP Typing (9 points)

Imagine a language where a derived class could indicate that one or more of the public methods defined in its base class will be converted to private methods in the derived class, e.g.:

```
class Animal {
public:
   virtual void eat() { ... }
   virtual void sleep() { ... }
   virtual void poop_in_public() { ... }   // ok for animals to do...
};

class Person: public Animal {
public:
  string get_name() const;
private:
  void poop_in_public() { ... }           // but not appropriate for people
};
```

This means that for example, if you created a Person object p in your main() function, you could not call p.poop_in_public(), because this method is private in Person.

```
int main() {
  Animal dog;
  dog.poop_in_public();       // compiles fine

  Person hercumur;
  hercumur.poop_in_public()  // compiler error
}
```

Part A (3 points)

Describe the typing relationship between the Animal class and the Person class, using terms like subtype, supertype, etc. Explain your reasoning.

Part B (3 points)

Assuming the language were statically typed, could we use polymorphism with these two classes? Explain your reasoning.

Part C (3 points)

What category of inheritance (that we learned in class) would be semantically most similar to this method-hiding approach, from the perspective of typing? Explain your reasoning.