# CS131 Fall 22 - Final Exam (Solutions)

# Dec 9th, 2022

This is a scantron test - you must place ALL answers on the scantron form in #2 pencil.

This exam is on double-sided paper.
Make sure to check for problems on both sides!

Do NOT write your answers on this exam.
This printed exam will NOT be collected or graded.
In fact, you may keep it.

Please practice academic integrity - don't cheat!

1. What is 10 + 10? (This question is not a joke. Please answer it correctly for an extra point.)

**A. 20**
B. 1000

2. The following program prints “Kitty’s name is Meomer” in some mystery language.

```
fn rename(Cat cat) {
   cat := new Cat(“Meomer”)
}

fn main() {
   kitty := new Cat(“Kittiekins”)
   rename(kitty)
   print(f”Kitty’s name is {kitty.name()}.”)
}
```

What type of parameter-passing semantics does this language use (choose one)?

A. This language uses pass-by-pointer.
B. This language uses pass-by-object-reference.
**C. This language uses pass-by-reference.**
D. This language uses pass-by-name.
E. It’s impossible to tell what parameter-passing scheme this language uses.

3. In the Okra language, we have a type called Carrot which can represent the values -4 to +4, and supports the addition, subtraction, and multiplication operations. We also have a type called Celery which can represent the values -4 to +8, and supports the addition, subtraction, multiplication and division operations. The language also has traditional integers, which have a range of -2 billion to +2 billion and support *all* traditional integer operations (e.g., addition, subtraction, multiplication, division, modulo, etc.). Given this information, which of the following are true (select all that apply):

A. Celery is a subtype of Carrot.
B. Carrot is a subtype of Celery.
**C. Neither Celery nor Carrot are subtypes of each other.**
D. Celery is a subtype of Okra integers.
E. Carrot is a subtype of Okra integers.

4. Given the following program:

```
fn bar() -> int {
  print(“bar “)
  return 5
}

fn foo(a int, b bool) -> unit {
  print(“foo “)
  if (b)
     print(f”{a} ”)
}

fn main() -> unit {
  foo(bar(), true)
  foo(bar(), true)
}
```

What would it print out if the language were using pass-by-name?

A. bar foo 5 bar foo 5
B. foo bar 5 foo 5
C. bar foo 5 foo 5
D. **foo bar 5 foo bar 5**
E. None of the above

5. We need to write a function to validate an SSL request packet that initiates an encrypted session with a website. If the request is valid, our function should return a valid encryption key, otherwise it will need to indicate what error occurred. The request may have several possible issues, including:

- The request may have an invalid format
- The request may ask for a deprecated encryption scheme
- The request may contain an encryption scheme not supported by the server

We expect that invalid requests will occur somewhat frequently (10% of the time), due to out-of-date browsers proposing the use of old encryption schemes, hackers, etc. Which of the following error-handling approaches would you use based on the criteria discussed in class?

A. Use assertions to validate each of the above items
B. Return an Optional object
**C. Return a Result object**
D. Raise an exception
E. Generate a panic

6. Consider the following Python comprehension:

```
x = [[1,2,3], [3,4,5],[5,6,7],[7,8,9,10]]
z = [[q[j] for q in x] for j in range(len(x[0]))]
```

What is the value of z after this code runs?

A. [1,2,3,3,4,5,5,6,7,7,8,9,10]
**B. [[1,3,5,7],[2,4,6,8],[3,5,7,9]]**
C. [[1,2,3],[3,4,5],[5,6,7],[7,8,9]]
D. [[1,3,5,7],[2,4,6,8],[3,5,7,9,10]]
E. None of the above

7. Consider the following Haskell comprehension:

```
c = [(a,b) | a <- [10..12], b <- ["a", "b"]]
```

What is the value of c?

A. [(10,"a"),(10,"b"),(11,"a"),(11,"b")]
B. [(10,"a"),(11,"a"),(10,"b"),(11,"b")]
**C. [(10,"a"),(10,"b"),(11,"a"),(11,"b"),(12,"a"),(12,"b")]**
D. [(10,"a"),(11,"a"),(12,"a"),(10,"b"),(11,"b"),(12,"b")]
E. None of the above

8. What result does the following Haskell expression produce:

```
foldr (\a b -> b-a) 3 (map (\a -> (-a)) [1,2,3])
```

A. An error: "Unexpected lambda expression in function application"
B. -5
C. 1
D. 3
E. **9**

For the next two questions, we will refer to the following code in Java. Java throws an ArithmeticException when you divide by zero:

```
void foo(int x) {
  if (x < 2) {
    try {
      int a = 5, c;
      System.out.println("B");
      c = a/x;
      System.out.println("C");
    } catch (NullPointerException e) {
      System.out.println("D");
      return;
    }
  }
  System.out.println("E");
}

void bar(int q) {
  try {
    System.out.println("A");
    foo(q);
    System.out.println("F");
  } catch (ArithmeticException e) {
    System.out.println("G");
  }
  finally { System.out.println("H"); }
  System.out.println("I");
}
```

9. What will be printed out if we call bar(0)?
   A. ABDI
   B. ABDHI
   C. ABDEHI
   D. **ABGHI**
   E. None of the above

10. What will be printed out if we call bar(1)?
   A. ABCEFH
   B. ABCDEFHI
   C. ABCGHI
   D. ABCEFI
   **E. None of the above**

11. The following program in a mystery language prints out 16:

```
function foo() {
  r = 5
  a = lambda(x) {
    print(x+r)
  }
  r += 1
  return a
}

f = foo()
f(10)
```

What are all of the capture strategies that are potentially consistent with the program's output (select all that apply)?

A. Capture by Value
**B. Capture by Reference**
**C. Capture by Environment**
D. Capture by Macro
E. None of the above

12. You have been given this excerpt of code from an unknown language, extracted from a broader class:

```
…
for obj in objects:
  obj.foo()
…
```

Which of the following statements are potentially true about the class that holds this code (select all that apply):

**A. The class that contains this code could be using duck typing.**
**B. The class that contains this code could be using subtype polymorphism.**
**C. The class that contains this code could be a generic class, but only if it's bounded.**
D. The class that contains this code could be a generic class, but only if it's not bounded.
**E. The class that contains this code could be using templates.**

13. Which of the following statements are NOT true (select all that apply):

A. An interface contains declarations for one or more related functions.
B. A function prototype is a declaration of a function.
**C. A function prototype is both a declaration and a definition of a function.**
D. An abstract class has one or more function declarations for which the functions lack definitions.
**E. A fully concrete class has at least one declaration that doesn't have an accompanying definition.**

14. Consider the following classes:

```
class Base {
  virtual void foo() { cout << "foo"; }
  virtual void bar() = 0;
  virtual void bletch() = 0;
};

class Derived: public Base {
  virtual void bar() { cout << "bar"; }
  virtual void bletch(string q) { cout << q; }
};
```

Select all the true statements about these two classes:

A. The definition of the Base class results in a value type.
B. **The definition of the Base class results in a reference type.**
C. The definition of the Derived class results in a value type.
**D. The definition of the Derived class results in a reference type.**
E. The Derived class has a single distinct type by which it can be referenced.

cc

15. What will the following code print out when executed:

```
class Foo {
public:
  Foo(int x) { s_  = x; }
  ~Foo() { s_ -= 1; }
  void print() const { cout << s_ << " "; }
private:
  static int s_;
};

int Foo::s_ = -10;

int main() {
  Foo a(1);
  a.print();
  Foo* p  = new Foo(2);
  a.print();
  p->print();
  Foo c(10);
  delete p;
  a.print();
  c.print();
}
```

A. 1 1 2 1 10
B. 1 2 2 1 9
C. 1 2 2 1 10
D. **1 2 2 9 9**
E. -10 2 2 9 9

16. You have two classes, a base class named Base and a derived class named Derived. *Importantly, Derived inherits from Base.*

You may assume that both classes compile just fine, without errors. However, when you pass an object of type Derived to a function that accepts a Base reference, the compiler gives you an error:

```
void process_base(Base& b) { … }

int main() {
  Derived d;
  process_base(d);  // compilation error here
}
```

Which of the following can you conclude about the above classes (select all that apply):

A. Subclassing must have been used when defining the Derived class.
B. Interface inheritance must have been used when defining the Derived class.
**C. Implementation inheritance must have been used when defining the Derived class.**
D. Prototypal inheritance must have been used when defining the Derived class.
E. The Derived class must be using composition and delegation.

17. Consider the following code in C++:

```
int a() { cout << “A”; return 0; }
int b() { cout << “B”; return 1; }
int c() { cout << “C”; return 2; }
int d() { cout << “D”; return 3; }
int e() { cout << “E”; return 4; }

int main() {
  if (a() > 10 && (b() > 5 || c() > 10) || d() < 10 || e())
    cout << “F”;
}
```

What does it print?

A. ABDEF
B. ADEF
C. ABCD
D. ABDF
**E. ADF**

18. What does this Python code print out?

```
def gen(lst):
 for i in lst:
  print(f"G {i}")
  yield i//10
 print("Done!")

lst = [10,20,30]
g = gen(lst)
print("Start!")
for i in g:
  print(f"M {i}")
  if i >= 0 and i < len(lst):
    lst[i] *= -1
```

A. Start! G 10 M 1 G 20 M 2 G 30 M 3 Done!
B. G 10 Start! M 1 G 20 M 2 G 30 M 3 Done!
C. G 10 Start! M 1 G -20 M -2 G 30 M 3 Done!
D. **Start! G 10 M 1 G -20 M -2 G 30 M 3 Done!**
E. None of the above

19. Which of the following statements are true?

**A. A list is an iterable object.**
**B. A range is an iterable object.**
C. An iterator is an iterable object.
D. A generator function is an iterator.
**E. A generator returns an iterator when it is called.**

20. Which of the following Prolog predicates unify with each other, given the provided initial mappings:

**A. Initial mapping = { }; foo(bar(X)) with foo(Y)**
**B. Initial mapping = { X → bletch }; foo(bar(X)) with foo(Y)**
C. Initial mapping = { Y → bletch }; foo(bar(X)) with foo(Y)
**D. Initial mapping = { }; foo(Q,R,c) with foo(a,b,G)**
E. Initial mapping = { }; foo(Q,Q,c) with foo(a,b,G)

21. Which of the following statements are true?

A. Prolog unification is the process of matching a query against all facts/rules in the prolog dataset to see if there is a match, and if so, extracting mapping(s).
B. **Prolog unification is the process of matching a query against one fact/rule to see if they match, and if so, extracting mapping(s).**
C. **Prolog resolution is the process of resolving a query against all facts/rules and obtaining either a boolean result or one or more mappings.**
D. Prolog resolution is the process of resolving a query against a single fact/rule to see if there is a match, and if so, extracting mapping(s).
E. Prolog unification leverages resolution to extract mappings.

22. Which of the following code snippets prepends an item onto a Prolog list? The predicate could be used as follows:

```
prepend_item([apple, beet, carrot], daikon, X)
```

and would yield:

```
X = [daikon, apple, beet, carrot]
```

A. prepend_item([],Q,[Q]).
prepend_item([Head|Tail],X,[Head|L]) :- prepend_item(Tail,X,L).
B. prepend_item([Q],[],[Q]).
prepend_item([Head|Tail],X,[X|Tail]) :- prepend_item(Tail,Head,X).
C. prepend_item([],Q,[Q]).
prepend_item(Q, X, [X | L | Tail]) :- prepend_item(Tail, X, L).
**D. prepend_item(Q, X, [X | Q]).**
E. prepend_item(Q, X, [Q | X]).

23. Consider the following code in a fictitious language:

```
func mystery(x) {
  var y = x;

  y = 2;
  print(f"y is {y}");
  return y;
}

var q = mystery(5.0);
print(f"q is {q}");
```

Which prints out the following:

```
y is 2.0
q is 2.0
```

What may you definitely conclude about this language (select all that apply)?

A. The language is dynamically typed.
B. **The language is statically typed.**
C. The language is using coercion in the call to the mystery function.
D. **The language is using coercion inside the mystery function.**
E. **The language is using type inference.**

24. Consider the following Haskell function which returns a new list with all consecutive duplicate values removed. It has some missing pieces labeled ??#??:

```
f [] = []
f [x] = ??0??
f (x : xs : ys)
    | ??1?? == ??2?? = f (??3??)
    | otherwise = ??4?? : f (??5??)
```

This function might be used as follows:

```
f [1,1,2,2,3,2,3]  → [1,2,3,2,3]
f [10] → [10]
f [] → []
```

What are the missing pieces?

A. ??0?? = [x], ??1?? = x, ??2?? = xs, ??3?? = xs:ys, ??4?? = x, ??5?? = x:ys
B. ??0?? = [xs], ??1?? = x, ??2?? = [], ??3?? = ys, ??4?? = xs, ??5?? = xs:ys
C. ??0?? = [], ??1?? = x, ??2?? = [xs], ??3?? = xs:ys, ??4?? = x, ??5?? = ys
D. ??0?? = [], ??1?? = x, ??2?? = ys, ??3?? = x:xs, ??4?? = xs, ??5?? = xs:ys
E. **??0?? = [x], ??1?? = x, ??2?? = xs, ??3?? = xs:ys, ??4?? = x, ??5?? = xs:ys**

25. Consider this Haskell function *foo* which accepts a function *g* and a list as parameters. It applies g to each integer in the list, and returns the sum of the overall result.

```
foo g [] = 0
foo g (x:xs) = g x + foo g xs
```

Which of the following is a valid type signature for foo?

A. foo::(Int -> (Int -> [Int] -> Int))
**B. foo::(Int -> Int) -> ([Int] -> Int)**
**C.** foo::Int -> (Int -> [Int]) -> Int
D. foo::([Int] -> Int) -> [Int] -> Int
E. None of the above

26. Given the *foo* function in Haskell below:

```
foo g [] = 0
foo g (x:xs) = g x + foo g xs
```

If we create a new function *bar* as follows:

```
bar = foo (\x -> 2*x)
```

What is the type signature of the bar function?

A. bar :: (Int -> Int) -> [Int] -> Int
B. bar :: (Int -> Int) -> Int
**C. bar :: [Int] -> Int**
**D.** bar :: Int -> Int
E. None of the above

27. Consider the following Haskell function called *del*, which searches for an item and deletes the first occurrence of that value from an ADT linked *List* of integers:

```
data List =
  Nil |
  Node Int List

del item Nil = Nil
del item (Node val next)
 | item == val = next
 | otherwise = (Node val (del item next))
```

If we called the *del* function on the following lists, how many new *Nodes* would be created during each of these three calls?

```
x = (Node 5 (Node 6 (Node 7 (Node 5 Nil))))
p = del 5 x
q = del 6 x
r = del 8 x
```

A. 4, 2, 0
B. 4, 4, 4
**C. 1, 2, 4**
D. 1, 2, 0
E. 4, 2, 5

28. Which of the following statements are true (choose all that apply):

**A. Coercions can happen in dynamically typed languages.**
**B. Static type checking may prevent some technically correct programs from compiling because static type checking is conservative.**
C. Passing a subtype object to a function that accepts a super-type object reference is using implicit conversion.
D. A language that throws an exception when an out-of-bounds array index occurs at runtime must be a weakly-typed language.
E. **Passing a subtype object to a function that accepts a super-type object reference is using implicit casting.**

29. Which of the following would be strictly associated with weakly-typed languages?

**A. Allowing a Dog object to be cast as a Robot**
B. Allowing an string containing an integer value, e.g., "42", to be implicitly converted into an integer in an operation like 5 * "42"
**C. Allowing an array to be indexed beyond its bounds**
D. Some objects are never be garbage collected even after they are no longer referenced
**E. Use of an uninitialized variable**

30. Which of the following goodies did Carey NOT hand out to reward participation this quarter (select all that apply)?

A. Mochi
**B. Toblerone**
C. Rice Krispie treats
D. Chips and Salsa
E. Diddy Riese