Feedback — Quiz #4

Help

You submitted this quiz on **Sun 27 Apr 2014 10:54 PM PDT**. You got a score of **10.00** out of **10.00**.

Question 1

In order, what are the COOL types for each of the three expressions in the following block? Assume a class hierarchy Foo $\leq$ Bar $\leq$ Bazz $\leq$ Object along with the usual primitive COOL types.

```
{
    case 3 of y: Int => y; z: Bool => z; esac;
    if 0 = 1 then new Bar else new Foo fi;
    let x: Bazz ← new Bar in x;
}
```

Your Answer		Score	Explanation
(•) Object, Bar, Bazz	✔	1.00	
( ) Int, Foo, Bazz			
( ) Object, Bazz, Bar			
( ) Int, Bazz, Bazz			
( ) Object, Foo, Bar			
( ) Int, Bar, Bazz			
( ) Object, Bar, Bar			
( ) Int, Foo, Bar			
( ) Object, Foo, Bazz			
( ) Int, Bazz, Bar			
Total		1.00 / 1.00	

Question Explanation

For expression 1, the type is lug(Int, Bool) = Object;
For expression 2, the type is lug(Bar, Foo) = Bar;
For expression 3, the type is static type of x = Bazz.

Question 2

If we have the following program:

```
Class Main inherits IO {
   x : Int ⟵ 5;
   foo(z:Int) : Int {
     x+z
   };
   bar(y:Int) : Int {{
     let x : Int ⟵ 1 in
     let z : Int ⟵ 2 in
     foo(y);
   }};
   main(): Object {{
     let x : Int ⟵ 7 in
     out_int(foo(bar(3)));
   }};
}
```

What would the the program print assuming it is statically scoped?

Your Answer		Score	Explanation
○ 11			
○ 15			
○ 9			
○ 5			
◉ 13	✔	1.00	
Total		1.00 / 1.00	

Question Explanation

If the program is statically scoped. In definition of function foo(), x binds to the attribute x defined before foo(); z binds to the parameter z. In the definition of function bar(), y binds to the parameter y. The result is 13.

Question 3

For the program in above question (Question 2), what would the the program print assuming it is dynamically scoped?

Your Answer		Score	Explanation
15			
9			
13			
11	✔	1.00	
5			
Total		1.00 / 1.00	

Question Explanation

If the program is dynamically scoped, any variable refers to the closest enclosing binding in the execution of the program.
When we call out_int(foo(bar(3)), we first compute foo(bar(3)) then output the return value.
When we call foo(bar(3)), we compute bar(3) then call foo().
When we compute bar(3), we call foo(3), x in foo() is binds to line let x:Int ←1in bar(), so foo(3) returns 4, then bar(3) return 4.
Now we call foo(4), now x in foo() binds to line let x:Int ←7 in main().So foo(4) returns 11.
That is the output of function out_int(11).

Question 4

Consider the following COOL program, with the type of method bar in class Bar omitted:

class Main {

```
  main() : Object {
    (new Bar).bar()
  };
};
class Foo inherits IO {
  foo() : SELF_TYPE {{
    out_string("Foo.foo()\n");
    foo();
    self;
  }};
};
class Bar inherits Foo {
  foo() : SELF_TYPE {{
    out_string("Bar.foo()\n");
    new SELF_TYPE;
  }};
  bar() : (*MISSING*) {
    case foo() of
      f : Foo => f@Foo.foo();
      b : Bar => (new Bazz).foo();
      o : Object => foo();
    esac
    };
};
class Bazz inherits Bar {
  foo() : SELF_TYPE {{
    out_string("Bazz.foo()\n");
    (new Bar)@Foo.foo();
    self;
  }};
};
```

Which of the following would be a valid return type for method bar in class Bar according to the static type checking rules for COOL?

[Check all that apply]

Your Answer		Score	Explanation
☑ Foo	✔	0.17	
☐ SELF_TYPE	✔	0.17	
☑ Object	✔	0.17	
☑ IO	✔	0.17	
☐ Bazz	✔	0.17	
☐ Bar	✔	0.17	
Total		1.00 / 1.00	

Question Explanation

According to static type checking rules in COOL, we need the (return value type) $\leq$(MISSING TYPE). And the return value type is lub(SELF_TYPE_Foo, SELF_TYPE_Bazz, SELF_TYPE_Bar). We know Bazz $\leq$ Bar $\leq$ Foo $\leq$ IO $\leq$ Object, so the MISSING TYPE could be Foo, IO or Object.
You can paste this program and test it using coolc.

Question 5

For the program in above question (Question 4), which is the output of the program?

Your Answer		Score	Explanation
○ Bar.foo(), Bar.foo()			
◉ Bar.foo(), Bazz.foo(), Foo.foo(), Bar.foo()	✔	1.00	
○ Bar.foo(), Foo.foo(), Bar.foo()			
○ Bar.foo(), Bazz.foo(), Foo.foo()			
○ Bar.foo(), Foo.foo(), Foo.foo(),... (infinite loop)			
○ Bar.foo(), Bazz.foo(), Foo.foo(), Bazz.foo()			
Total		1.00 / 1.00	

Question Explanation

The main() calls (new Bar).bar(), which will call bar() defined in class Bar.
When executing the function and evaluating the case expression, the program first calls function foo() defined in class Bar.
The program outputs "Bar.foo()", then returns a new Bar object.
So in the case expression, the program continues to evaluate the b: Bar=>(new Bazz).foo(). Now the program calls foo() defined in Bazz class.
The program outputs "Bazz.foo()", and then calls the function foo() defined in class Foo(static dispatch here).
The program outputs "Foo.foo()", then calls foo() defined in class Bar which outputs "Bar.foo()".
So the output of the program is Bar.foo() Bazz.foo() Foo.foo() Bar.foo().

Question 6

Consider the following COOL program:

```
class Main inherits IO {
    sing() : Int { 1 };
    valcomp(x: Int, y:Int) : Bool { x ≤ y };
    fib(x:Int) : Int {
        if valcomp(x,2) then sing() else fib(x - 1) + fib(x - 2) fi
 };
    main() : Int {{
        x:Int ← let y : Int ← 4 in fib(y);
        out_int(x);
     }};
};
```

In the activation tree, how many direct children does the activation of the first call to fib in main have, and what is the maximum depth of the activation tree? Assume a single root node as a tree has a a depth 1.

Your Answer	Score	Explanation
○ Children: 2, Depth: 3		
○ Children: 2, Depth: 4		
○ Children: 3, Depth: 6		

Children: 4, Depth: 4		
Children: 4, Depth: 5		
Children: 2, Depth: 5		
Children: 3, Depth: 5	✔	1.00
Total		1.00 / 1.00

Question Explanation

The activation tree is like following.
main() is the root node and has two children, fib(4) and out_int().
 fib(4) has 3 children, valcomp(), fib(3) and fib(2).
  valcomp() has no children.
  fib(3) has 3 children, valcomp(), fib(2) and fib(1).
   valcomp() has no children.
   fib(2) has 2 children, valcomp() and sing(), both of which have no children.
   fib(1) has 2 children, valcomp() and sing(), both of which have no children.
  fib(2) has 2 children, valcomp() and sing(), both of which have no children.
 out_int has no children.

Question 7

For the program in above question (Question 6), how many activation records are there in total, assuming out_int has no child calls?

Your Answer		Score	Explanation
17			
12			
21			
10			
15	✔	1.00	
8			
Total		1.00 / 1.00	

Question Explanation

See the activation tree in last question's explanation.

Question 8

For the program in above question (Question 6), which are possible activation stack states during the execution of main?

The answer from left to right represents from the top to the bottom of the stack.

[Check all that apply.]

Your Answer		Score	Explanation
☐ out_int;fib;main	✔	0.17	
☑ sing;fib;fib;fib;main	✔	0.17	
☑ sing;fib;fib;main	✔	0.17	
☑ main	✔	0.17	
☐ main; out_int	✔	0.17	
☐ sing;valcomp;fib;main	✔	0.17	
Total		1.00 / 1.00	

Question Explanation

You can simulate the execution of the program and get the answer.

Question 9

Consider the expression (7 + 5) * (3 + 2).

Which of the following are possible stack machine states during evaluation?

[Check all that apply]

Your Answer		Score	Explanation
☐ Acc: 5, Stack: 7, 5, ＜init＞	✔	0.17	

☐ Acc: 12, Stack: 5, <init>	✔	0.17
☑ Acc: 3, Stack: 12, <init>	✔	0.17
☐ Acc: 5, Stack: <init>	✔	0.17
☑ Acc: 5, Stack: 7, <init>	✔	0.17
☐ Acc: 3, Stack: 2, 7, 5, <init>	✔	0.17
Total		1.00 / 1.00

Question Explanation

The code to evaluate the expression is:
acc ← 7; push acc; acc ← 5; acc ← acc + top_of_stack; pop; push acc; acc ← 3; push acc; acc ← 2; acc ← acc + top_of_stack; pop; acc ← acc * top_of_stack; pop.

Question 10

Consider computing expressions using a 1-register stack machine.

[Check all that apply.]

Your Answer		Score	Explanation
☑ Computing expressions 1+2+3+...+n and 1+(2+(3+(...+n))) require the same number of acc ← acc + top_of_stack actions.	✔	0.20	
☑ Computing expressions 1+2+3+...+n and 1+(2+(3+(...+n))) require different amounts of stack space.	✔	0.20	
☑ Conputing the expression 1+2+3+..._n requires n-1 push actions.	✔	0.20	
☐ Computing the expression 1+2+3+...+n requires 1 push action.	✔	0.20	
☐ Computing the expression 1+(2+(3+(...+n))) requires n pop actions.	✔	0.20	
Total		1.00 / 1.00	

Question Explanation

Computing expression 1+2+3+...+n requires n-1 push actions,n-1 pop actions and n-1 acc ← acc + top_of_stack actions. There is at most 1 integer in the stack.
Computing expression 1+(2+(3+(...+n))) requires n-1 push actions, n-1 pop actions and n-1 acc ← acc + top_of_stack actions. There are at most n-1 integers in the stack.