

14. Consider these two definitions of classes in Java. The objects can be used to represent a list of integers, assuming that “n” is an integer. They are a form of class.

```
interface List ...;
class NIL implements List {
    public isEmpty() { return true; }
    public current() { throw new Error(); }
    public next()    { throw new Error(); }
};
class CONS implements List {
    int n; List l;
    public CONS(int n, List l) { this.n = n; this.l = l; }
    public bool isEmpty() { return false; }
    public int  current() { return n; }
    public List next()    { return l; }
};
List l = new CONS(4, new CONS(3, new NIL()));
```

- a) Give an appropriate definition for the interface List;

```
interface List {
    bool isEmpty();
    int  current();
    List next();
};
```

- b) Implement another class named POWERS that takes an integer n as an input and returns an object representing the infinite list containing the powers of 2. Note that if the input  $n = 2^i$  then the next number in the sequence is  $2^{i+1}$ . That is, given

```
List from2 = new POWERS(2);
int x = from2.next().next().current();
Then x should be equal to  $2^3 = 8$ .
```

```
class POWERS implements List {
    int n;
    public POWERS(int n) { this.n = n; }
    public bool isEmpty() { return false; }
    public int  current() { return n; }
    public List next()    { return new POWERS(n * 2); }
};
```

1. How many cons cells will exist right after the garbage collection and before the call to “equal”? Explain for partial credit.

```
(define input (cons 6 (cons 12 (cons 19 nil)))) ; hint: input has 3 cons cells
(define result1 (cons (cdr input) (cons 15 input)))
(define result2 (cons (car (cons 5 input)) (cons 15 input)))
; garbage collection happens here. input, result1 and result2 are still defined
(equal result1 result2)
```

**A: there are 8 cons cells created, and one is garbage: the one created by (cons 5 input). So 7 are left after garbage collection.**

15. The intent of the following GridPoint class is to ensure that the getX() message always return an integer that is a multiple of “g”. Describe whether this is true and what you would do to fix any problems in the code.

```
class Point {
    private int x;
    public Point(int nx) { x = nx; }
    int getX() { return x; }
    void setX(int nx) { x = nx; }
```

```

void move(int dx) { x += dx; }
bool compare(Point other) { return (x == other.getX()); }
}
class GridPoint extends Point {
  int g;
  public GridPoint(int nx, int ng) { super(nx); g = ng; }
  void setX(int nx) { super.setX(Math.round(nx / g) * g); }
}

```

**A: The move method in Point changes x but it is not necessarily a multiple of g. Change this line:**

```
void move(int dx) { setX(x + dx); }
```

16. Consider the following abstract data type for sets:

```

abstype set = EMPTY | ELEM of int * set with
  val empty = EMPTY
  fun insert(x, s) = ELEM(x, s)
  fun isMember(x, s) =
    let fun find(e, EMPTY) = false
        |   find(e, ELEM(x, s)) = if e=x then true else find(e, s)
    in
      find(x, s)
    end
end;

```

a) What is the interface of this abstract type?

```

abstype set
  val empty : set
  fun insert : int * set -> set
  fun isMember : int * set -> bool

```

b) Is the following client program legal? Explain:

```

val q = insert(4, insert(2, empty));
fun size(EMPTY) = 0
|   size(ELEM(x, s)) = 1 + size(s);
val r = size(q);

```

**A: no, can't see the representation fields EMPTY, ELEM outside the abstract type definition.**

17. (20 points) Consider the following RECORD types:

```

type A = { name: string, age: int }
type B = { name: string, weight: float }
type C = { age: int, weight: float }
type D = { name: string }

```

a) List all the subtyping relationships between these types. Use the notation  $X <: Y$  for X subtype Y.

$A <: D, B <: D$

b) Write down a type that is a subtype of all these types

```
{ name: string, age: int, weight: float }
```

18. (20 points) Consider the following types:

```

type A = {x: int, y: int } → { name: string }
type B = {x: int } → { age: int, name: string }
type C = {x: int, y: int, z: int } → { age: int, name: string }

```

c) List all the subtyping relationships between these types

$B <: A, B <: C$

d) Write down a type that is a subtype of all these types

```
{x: int } → { age: int, name: string }
```