1. Contracts

The deferred class **Time** in the following code abstracts a simple interface of time in 24-hour format; the range of valid time is specified by the class invariant. Please specify the proper preconditions/postconditions for the class routines.

```
deferred class TIME

feature -- Initialization

    make (a_hour, a_min, a_sec: NATURAL_8)
            -- Initialize ‘hour’, ‘minute’, and ‘second’ with ‘a_hour’,
            -- ‘a_min’, and ‘a_sec’, respectively.
        require

         ..................................................................

         ..................................................................

         ..................................................................
        do
            set_hour (a_hour); set_minute (a_minute); set_second (a_second)
        ensure

         ..................................................................

         ..................................................................

         ..................................................................
        end

feature -- Access

    hour: NATURAL_8 assign set_hour
            -- Hour.

    minute: NATURAL_8 assign set_minute
            -- Minute.

    second: NATURAL_8 assign set_second
            -- Second.

feature -- Setters

    set_hour (a_hour: NATURAL_8)
            -- Set `hour' to be ‘a_hour’.
        require

         ..................................................................
        do
            hour := a_hour
        ensure

         ..................................................................
        end
```

```
    set_minute (a_minute: NATURAL_8)
            -- Set `minute' to be ‘a_minute’.
        require -- omitted here
        do
            minute := a_minute
        ensure -- omitted here
        end

    set_second (a_second: NATURAL_8)
            -- Set `second' to be ‘a_second’.
        require -- omitted here
        do
            second := a_second
        ensure -- omitted here
        end

feature -- Operation

    tick
            -- Tick the time to the next second, in 24-hour format.
            -- During a day, “tick” works in the usual way. For example,
            -- the next second of 07:28:59 would be 07:29:00. While
            -- the next second of 23:59:59 would be 00:00:00.
        deferred
        ensure

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................
        end

invariant

     hour_in_range: hour < 24

     minute_in_range: minute < 60

     second_in_range: second < 60
end
```

2. Void Safety

The void safety mechanism prevents runtime "Call on void target." errors by ensuring that: (1) A statically attached reference is always dynamically attached at run-time after initialization; and (2) A call *x.f(…)* is only permitted if *x* is a statically attached reference.

Read the following program and answer the questions on the following page, assuming that the compiler enforces void safety. (*foo* is a feature in class **A**.)

```
deferred class C

feature

    attribute_one: detachable A

    operation_one (a_arg1: detachable A; a_arg2: attached A)
        local
            l_loc1: attached A
            l_loc2: detachable A
        do
            l_loc1 := a_arg1                -- (i)
            l_loc2 := a_arg2                -- (ii)
        end

    operation_two (a_arg: detachable A)
        do
            if a_arg /= Void then
                operation_four
                a_arg.foo                   -- (iii)
            end
        end

    operation_three (a_arg: detachable A)
        do
            attribute_one := a_arg
            if attribute_one /= Void then
                operation_four
                attribute_one.foo           -- (iv)
            end
        end

    operation_four
        deferred
        end

end
```

(1) Are the assignments on line (i) and (ii) valid? Why or why not?

(2) Are the feature calls on line (iii) and (iv) valid? Why or why not? If not, please correct the code to make it valid (with as small a change as possible) while preserving the intended semantics.

3. Programming

Skip lists are a data structure for storing sorted items. In simple terms, skip lists are sorted linked lists with two differences:

1. **Forward references.** The nodes in ordinary lists have one *next reference*, while the nodes in skip lists may have multiple "next" references – called *forward references*. A node *always* has a forward reference pointing to its next node, and some nodes have extra references to nodes along the list. When the list is traversed, these extra references allow some nodes to be "skipped".

2. **Node size.** The number of forward references of a node is called the *size* of the node. The size of a node is at least ***1*** (according to 1) and at most ***max_size***. The header node and the tail node always have the size ***max_size***, and the sizes of other nodes are determined probabilistically. Given a fixed probability ***p***, the probabilities for a node having size ***2***, ***3, 4*** …, are ***p***, p^2, p^3… (***0*** <= ***p*** <= ***1***), respectively.
The maximum node size ***max_size*** and the probability ***p*** are two characteristics of a skip list, and they are both fixed at the construction time.

As shown by the example in Figure 1, a skip list can be conceptually understood as consisting of several layers and each layer constitutes a sorted linked list containing a subset of the items. Notice that a node of size ***n*** is linked into layers ***1 – n***.

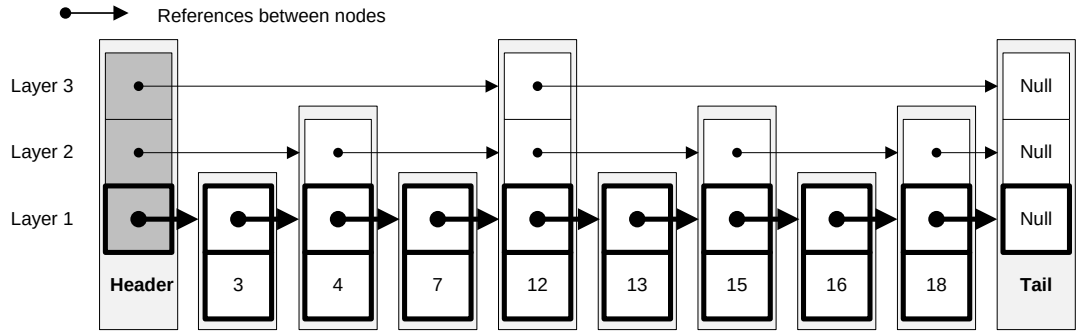

Figure 1. Layered structure of a skip list

Inside a skip list, we need both a skip list node and a layer to specify a *reference position*. In the following, we use the value stored at a node to denote that node, and use a pair ***(node, layer)*** to denote a reference position in a skip list. For example, reference at position ***(12, 2)*** is the reference at node ***12*** in layer ***2***. The *value* at a reference position is determined exclusively by the node at that position. Thus, the value at position ***(12, 2)*** would be ***12***.

When traversing a skip list, we accordingly can move in two directions from a reference position. We can either move to the lower layer of the current node, or move to the next node in the same layer by following the reference. For example, we can move from ***(12, 2)*** to either ***(12, 1)*** (called *downward position*) or ***(15, 2)*** (called *forward position*).

Given the skip list shown in Figure 1, the diagram in Figure 2 illustrates how the *search* and *insert* operations could be done with efficiency.

- To search for value ***16*** in the skip list, we start from position ***(header, 3)***. Since the value at the forward position is ***12***, which is smaller than ***16***, we move forward to position ***(12, 3)***; now the forward position is at the tail node, so we move downward to ***(12, 2)***; the value at the forward position is now ***15***, which is still smaller than ***16***, so we move forward again to position ***(15, 2)***; here the forward position has value ***18***, which is bigger than ***16***, thus we move downward to ***(15, 1)*** and then ***16*** is found at its forward position.
- To insert a value, say ***17***, into a skip list, we first create a node to accommodate the value ***17***. Suppose the node creation procedure decides (probabilistically) that the new node has size ***3***, we then collect the *last* positions with values smaller than ***17*** in layers ***1 – 3***. Here we (only) need to do this in layers ***1 – 3*** is because, as mentioned earlier, a node of size ***n*** is (only) linked into layers ***1 – n***. The references at the collected positions will point to the inserted node after insertion, and, in this example, these positions include ***(12, 3)***, ***(15, 2)***, and ***(16, 1)***. After that, we can update accordingly the references at the collected positions and in the new node to link the new node into the skip list, in all ***3*** layers.

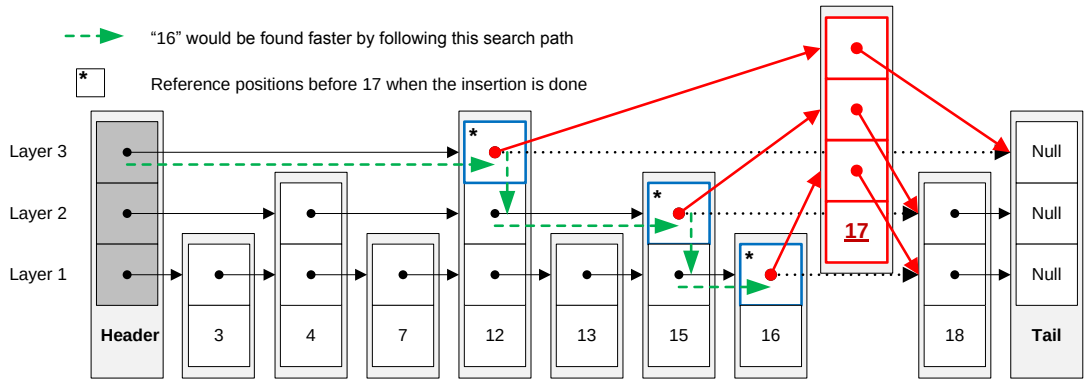

Figure 2. Search and insertion in a skip list

A partial implementation of a skip list has been given below. The class **DS_SKIP_LIST_NODE** is already complete. Its creation feature makes use of a random number generator to determine the size of the new created node.

The elements in the skip list are sorted in ascending order. Please provide the missing part of the feature *insert* in class **DS_SKIP_LIST** so that, when called, it inserts its argument *a_value* into the list, if *a_value* is not already in the list.

(In case you need the interface information for class **ARRAY** and **COMPARABLE**, it is shown on the next page.)

```
class ARRAY[G]

feature

    lower: INTEGER
            -- Minimum index.

    upper: INTEGER
            -- Maximum index.

    count: INTEGER
            -- Number of available indexes.

    item (i: INTEGER): G
            -- Entry at index `i'.

    put (v: like item; i: INTEGER)
            -- Replace `i'-th entry, if in index interval, with `v'.

    ...
end
```

```
deferred class COMPARABLE

feature -- Comparison

    is_less alias "<" (other: like Current): BOOLEAN
            -- Is current object less than `other'?

    is_less_equal alias "<=" (other: like Current): BOOLEAN
            -- Is current object less than or equal to `other'?

    is_greater alias ">" (other: like Current): BOOLEAN
            -- Is current object greater than `other'?

    is_greater_equal alias ">=" (other: like Current): BOOLEAN
            -- Is current object greater than or equal to `other'?

    is_equal (other: like Current): BOOLEAN
            -- Is `other' attached to an object of the same type
            -- as current object and identical to it?

    ...
end
```

```
class
    DS_SKIP_LIST_NODE [G -> COMPARABLE]

create
    make_empty, make_with_value

feature{DS_SKIP_LIST} -- Initialization

    make_empty (a_no_of_layers: NATURAL_8; a_probability: REAL)
            -- Initialize a skip list node.
        require
            a_no_of_layers_big_enough: a_no_of_layers >= 1
            a_probability_in_range:
                0 <= a_probability and a_probability <= 1
        local
            l_layers: NATURAL_8
            l_continue: BOOLEAN
            l_random: RANDOM
        do
                -- compute probabilistically the size of the new node
            from
                l_random := Rand
                l_layers := 1
                l_continue := True
            until
                not l_continue or l_layers >= a_no_of_layers
            loop
                l_random.forth
                if (l_random.real_item <= a_probability) then
                    l_layers := l_layers + 1
                else
                    l_continue := false
                end
            end
            check 1 <= l_layers and l_layers <= a_no_of_layers end

            size := l_layers
            create links.make_filled (Void, 1, l_layers)
        ensure
            size_in_range: 1 <= size and size <= a_no_of_layers
            links_not_void: links /= Void
        end

    make_with_value (a_no_of_layers: NATURAL_8; a_probability: REAL;
                     a_value: G)
            -- Initialize a skip list node and set 'value' to be 'a_value'.
        require
            a_no_of_layers_big_enough: a_no_of_layers >= 1
            a_probability_in_range: 0 <= a_probability and a_probability <= 1
        do
            make_empty (a_no_of_layers, a_probability)
            set_value (a_value)
        ensure
            size_in_range: 1 <= size and size <= a_no_of_layers
            links_not_void: links /= Void
```

```
            value_set: value = a_value
        end

feature{DS_SKIP_LIST} -- Access

    size: NATURAL_8
            -- Number of forward references in the node.

    links: ARRAY[detachable DS_SKIP_LIST_NODE[G]]
            -- List of forward references.

    value: detachable G assign set_value
            -- Value.

feature{DS_SKIP_LIST} -- Setting

      set_value (a_value: detachable G)
                  -- Set the value to be 'a_value'.
             do
                 value := a_value
             ensure
                 value_set: value = a_value
             end

feature{NONE} -- Implementation

      Rand: RANDOM
                  -- Random number generator.
             once
                 create Result.make
             end

end
```

```
class
    DS_SKIP_LIST [G -> COMPARABLE]

create
    make

feature{ANY} -- Initialization

    make (a_no_of_layers: NATURAL_8; a_probability: REAL)
            -- Initialize a skip list.
        require
            no_of_layers_big_enough: a_no_of_layers >= 1
            probability_in_range: 0 <= a_probability and a_probability <= 1
        local
            l_count: NATURAL_8
        do
            number_of_layers := a_no_of_layers
            probability := a_probability

                -- make sure the tail node has maximum size
            create tail.make_empty (a_no_of_layers, 1.0)
```

```
            -- make sure header has maximum size
        create header.make_empty (a_no_of_layers, 1.0)

            -- initially, all forward references in the header node
            -- point to the tail node
        from
            l_count := 1
        until
            l_count = a_no_of_layers + 1
        loop
                -- note that references actually point to nodes
            header.links.put (tail, l_count)
            l_count := l_count + 1
        end
    end

feature -- Access

    probability: REAL
            -- Fixed probability for each skip list, which decides
            -- the distribution of node size.

    number_of_layers: NATURAL_8
            -- Fixed total number of layers.

    header: DS_SKIP_LIST_NODE[G]
            -- List header.

    tail: DS_SKIP_LIST_NODE[G]
            -- List tail.

feature{ANY} -- Operation


    insert (a_value: G)
            -- Insert `a_value' into the list.
        require
                -- Suppose there is a feature `has'
                -- checking if `a_value' is already in the skip list
            not_in_list: not has (a_value)
        local

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................

         ..................................................................
```

.....

.....

.....

.....

.....

.....

.....

do

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
          ..........................................................................
        ensure
            value_inserted: has (a_value)
        end

            -- Other implementation details omitted
        ......
end
```

Please provide the missing part of the feature *has* in class DS_SKIP_LIST so that, when called, it checks whether its argument *a_value* is already in the list.

```
    has (a_value: attached G): BOOLEAN
            -- Is `a_value' in the list?
        local
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
        do
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
         .....................................................................
```

```
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
  .....................................................................
end
```