ETHZ D-INFK Final Exam Booklet

Eiffel: Analysis, Design and Programming

Date: 15 December 2009

Surname, first name: ....

Student number: ....

I confirm with my signature, that I was able to take this exam under regular circumstances and that I have read and understood the directions below.

Signature: ....

Directions:

- Exam duration: 90 minutes.
- Except for a dictionary you are not allowed to use any supplementary material.
- Use a pen (**not** a pencil)!
- All solutions can be written directly on the exam sheets. If you need more space for your solution ask the supervisors for a sheet of official paper. You are **not** allowed to use other paper. Please write your student number on **each** additional sheet.
- Only one solution can be handed in per question. Invalid solutions need to be crossed out clearly.
- Please write legibly! We will only correct solutions that we can read.
- Please **immediately** tell the exam supervisors if you feel disturbed during the exam.

Good luck!

Question	Number of possible points	Points
1	15	
2	20	
3	20	
4	15	
5	30	
Total	100	

1. Contracts (15 Points)

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 pre-conditions/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   -- (3 Points)

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

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

         ......................................................................
        do
            set_hour (a_hour); set_minute (a_minute); set_second (a_second)
        ensure   -- (3 Points)

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

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

         ......................................................................
        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   -- (1 Point)

         ......................................................................
        do
            hour := a_hour
        ensure   -- (1 Point)

         ......................................................................
        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   -- (7 Points)

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

invariant

    hour_in_range: hour < 24

    minute_in_range: minute < 60

    second_in_range: second < 60
end
```

2. Program Comprehension (20 Points)

The following program with root class **APPLICATION** and root creation procedure **make** uses multiple inheritance. Read the program and answer the questions that follow it.

```eiffel
class APPLICATION

create
    make

feature
    make
        local
            a: A
            c: C
            d: D
            l_retried: BOOLEAN
        do
            if not l_retried then
                create c
                create d

                a := c
                a.f

                a := d
                a.f
                c := d
                c.k
                d.g
            end
        rescue
            l_retried := True
        end
end
```

```eiffel
class A

create
    default_create

feature
    f
        do
            io.put_string ("A.f%N")
        end

    g
        do
            io.put_string ("A.g%N")
                -- Attention: we call f here
            f
        end
```

```
end
```

```
deferred class B

inherit
    A
        rename f as h
        undefine g
        redefine h
        end

feature

    h
        do
            io.put_string ("B.h%N")
        end
end
```

```
class C

inherit
    A
        rename f as k
        redefine g
        end

    EXCEPTIONS

create
    default_create

feature
    g
        do
            Precursor
            io.put_string ("C.g%N")
            raise ("An error")
        end
end
```

```
class D

inherit
    B
    .......................................    -- what's missing here?

    C

create
    default_create

end
```

Questions:

1. To make the program compile successfully, what should be added on the dotted line in class **D**? Why? (write your answer below) (6.5 points)

   ....

   ....

   ....

   ....

   ....

   ....

   ....

   ....

   ....

2. Please write below the <u>first 5 lines</u> of the console output when the program (corrected per question 1) is executed, <u>in output order</u>. (1.5 point × 5)

   ```
   Line (1)  ..............................................
   Line (2)  ..............................................
   Line (3)  ..............................................
   Line (4)  ..............................................
   Line (5)  ..............................................
   ```

3. Will the program terminate with or without an exception? Why?
   (write your answer below) (6 points)

   ....

   ....

   ....

   ....

   ....

   ....

   ....

   ....

3. Multiple Choice (4 Points × 5)

Read the questions below and write down your selections in the following table. <u>Only answers written in the table will be graded!</u>

Question	1	2	3	4	5
Selection(s)					

1. Which of the following descriptions about the EiffelStudio compilation process, if any, is/are <u>correct</u>?

    A. The *Melting Ice Technology* is introduced to make compilation of large systems during development more efficient.

    B. During melting, only Eiffel code that has been changed will be recompiled into C code.

    C. During execution of a system with both melted and frozen code, the melted code will be interpreted while the rest is executed directly.

    D. Finalizing is a non-incremental process of producing efficient C code and the resulting executable.

2. Given the conforming inheritance relation between class **A** and **D** as shown in Figure 1, f is a feature in **A** with precondition P_f, postcondition Q_f, and a single argument of type T_A, and the redefined version of f in **D** has precondition $P_f{}'$, postcondition $Q_f{}'$, and argument type T_D. Which of the following statements (if any) is/are <u>correct</u>?

    A. $P_f{}'$ should be weaker than or equal to, P_f;

    B. $Q_f{}'$ should be weaker than or equal to, Q_f;

    C. T_D should conform to T_A;

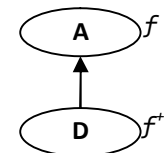   D. None of the above is correct.

Figure 1. Inheritance between **A** and **D**

3. Let $is\_greater$ be a feature in class **C**, with signature
    ```
    is_greater (an_int1, an_int2: INTEGER): BOOLEAN
    ```
    Which of the following local variable declarations and statements, if any, are <u>valid</u> when used together? Suppose c is an attached reference with type **C** and l_bool a boolean local variable.

    A.
    ```
    l_agent: PROCEDURE[ANY, TUPLE[INTEGER], BOOLEAN]
    l_agent := agent c.is_greater(?, 10)
    l_bool := l_agent.call (4)
    ```

    B.
    ```
    l_agent: FUNCTION[ANY, TUPLE[INTEGER], BOOLEAN]
    l_agent := agent c.is_greater(?, 10)
    l_bool := l_agent.item([4])
    ```

C. `l_agent: FUNCTION[C, TUPLE[INTEGER, INTEGER], BOOLEAN]`
`l_agent := agent c.is_greater(?, 10)`
`l_bool := l_agent.call([4])`

D. `l_agent: PROCEDURE[C, TUPLE[ ], BOOLEAN]`
`l_agent := agent c.is_greater(?, 10)`
`l_bool := l_agent.call (4)`

4. Given the following local variable declarations, with class **D** inherits from class **A** as shown in Figure 1,

```
l_a: A
l_d: D
l_t1: TUPLE[INTEGER]
l_t2: TUPLE[INTEGER, INTEGER]
```

which of the assignments below, if any, is/are <u>correct</u>?

A. `l_a := l_d`

B. `l_d := l_a`

C. `l_t1 := l_t2`

D. `l_t2 := l_t1`

5. The C function *strncmp* compares up to the first *n* bytes of two strings lexicographically, and its declaration in string.h is as follows:

`int strncmp(const char * str1, const char * str2, size_t n);`

To wrap this function *directly* into an Eiffel feature *strncmp*, which of the following signatures, if any, is/are valid for feature *strncmp*?

A. `strncmp (p1, p2: POINTER; count: INTEGER): INTEGER`

B. `strncmp (p1, p2: POINTER; count: SIZE_T): INTEGER`

C. `strncmp (str1: EIF_POINTER; str2: EIF_POINTER; n: INTEGER): INTEGER`

D. `strncmp (str1: EIF_POINTER; str2: EIF_POINTER; n: SIZE_T): INTEGER`

4. Void Safety (15 Points)

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_three
                a_arg.foo                  -- (iii)
            end
        end

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

    operation_three
        deferred
        end

end
```

(1) Are the assignments on line (i) and (ii) valid? Why or why not? (6 Points)

(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. (9 Points)

5. Programming (30 Points)

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 2, 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*.
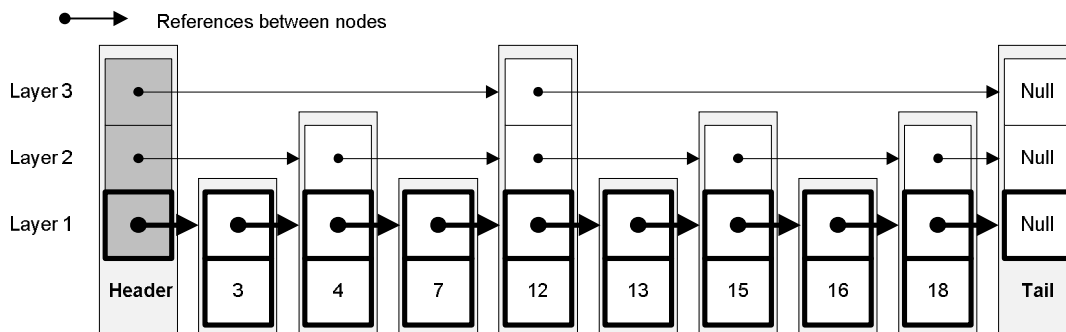
Figure 2. Layered structure of a skip list

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 2, the diagram in Figure 3 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.
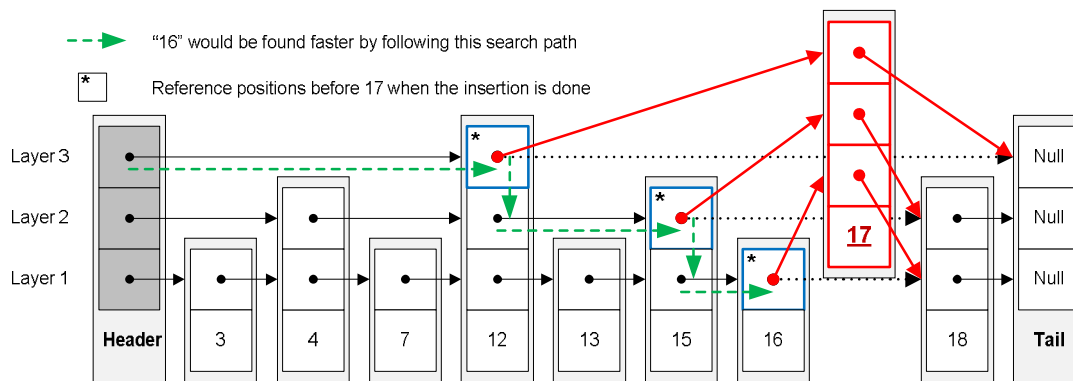
Figure 3. Search and insertion in a skip list

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)
```

```eiffel
                -- 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. (30 Points)
        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

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

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

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

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

6. Bonus Question (20 Points)

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