Introduction to Programming

ETH Zurich

Date: 19.08.2013

Family name, 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: 120 minutes.
- Except for a dictionary you are not allowed to use any supplementary material.
- Use a pen (**not** a pencil)!
- Please write your student number onto **each** sheet.
- All solutions can be written directly onto 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.
- 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.
- Manage your time carefully (take into account the number of points for each question).
- Please **immediately** tell the exam supervisors if you feel disturbed during the exam.

Good luck!

Question	Number of possible points	Points
1	10	
2	14	
3	12	
4	13	
5	16	
Total	65	

1 Multiple choice (10 points)

Put checkmarks in the checkboxes corresponding to the correct statements. There is at least one correct answer per question. A correctly checked or unchecked box is worth 0.5 points. An incorrectly checked or unchecked box is worth 0 points. Completely unanswered questions are worth 0 points.

Example:

Which of the following statements are true?

a. The sun is a mass of incandescent gas. ☒ 0.5 points
b. $2 \times 4 = 8$ ☐ 0 points
c. "Rösti" is a kind of sausage. ☒ 0 points
c. C is an object-oriented programming language. ☐ 0.5 points

---

1. Objects and classes.
   a. A program can create an unlimited number of objects of a certain type. ☐
   b. Creation procedures can be made accessible to only a limited number of clients. ☐
   c. In Eiffel any function can be a creation procedure as well. ☐
   d. A generic class *C* [*G*] represents one type. ☐
   e. The public interface of a class includes all the features that are available to all clients. ☐

2. Control structures and recursion.
   a. If we know that a loop decreases its variant and that it never goes below 5, then we know that the loop terminates. ☐
   b. The loop invariant may be violated during the loop initialization (before entering the loop itself). ☐
   c. The loop invariant tells us how many times the loop will be executed. ☐
   d. In Eiffel a procedure is always allowed to call itself on the same object. ☐
   e. A loop can always be rewritten as a finite sequence of conditional statements and compound statements. ☐

3. Inheritance and polymorphism.
   a. All classes in Eiffel implicitly inherit from class *OBJECT*. ☐
   b. At runtime a variable can be attached to an object, whose dynamic type inherits from the variables's static type. ☐
   c. At runtime a variable can be attached to an object, whose dynamic type is the same as the variables's static type. ☐
   d. At runtime a variable can be attached to an object, whose dynamic type is an ancestor of the variables's static type. ☐
   e. For an object *obj*, the feature call *obj.is_equal*(*obj*) can return **False**. ☐

4. Design by Contract.
   a. In a creation procedure the class invariant of the created object holds at the beginning and at the end of the procedure body. ☐
   b. A class invariant can be violated without consequences in the body of a routine. ☐
   c. The invariant provided by a class is and-ed with those of its ancestors. ☐
   d. A function redefining another function can provide a precondition that is stronger than the one given by the original function. ☐
   e. A procedure redefining another procedure can provide a postcondition that is weaker than the one given by the original procedure. ☐

2 Inheritance and Polymorphism (14 Points)

Classes *SCIENTIST*, *COMPUTER_SCIENTIST*, *BIOLOGIST*, and *PET* shown below are part of an application for managing scientists' social life on the web.

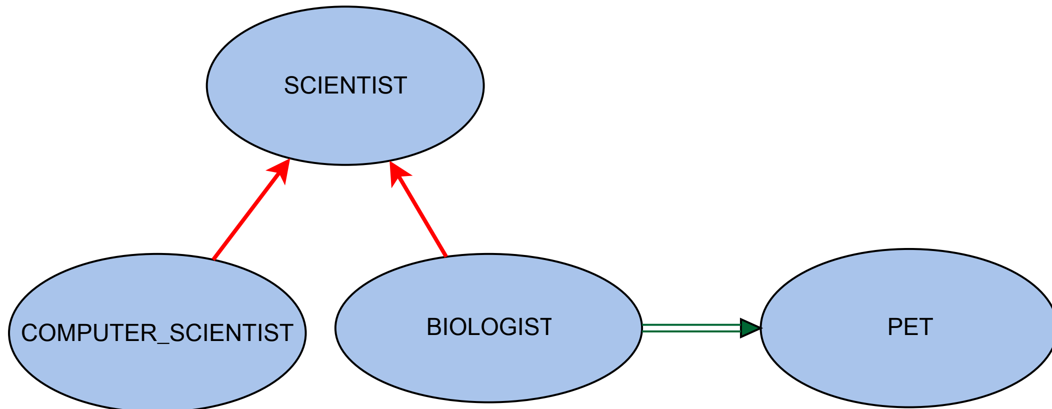

Figure 1: BON Diagram

```
deferred class
	SCIENTIST

feature {NONE} -- Initialization

	make (a_name: STRING)
			-- Initialize Current with 'a_name'.
		require
			a_name_exists: a_name /= Void and then not a_name.is_empty
		do
			name := a_name
		ensure
			name_set: name = a_name
		end

feature -- Access

	name: STRING
			-- Current's name.

feature -- Basic operations

	introduce
			-- Print info about self.
		do
			io.put_new_line
			print ("My name is " + name + "; ")
		end
end
```

```
class
  COMPUTER_SCIENTIST

inherit
  SCIENTIST
    redefine
      introduce
    end

create
  make

feature -- Basic operations

  introduce
      -- Print info about self.
    do
      Precursor
      print ("I am a computer scientist.")
    end
end
```

```
class
  BIOLOGIST

inherit
  SCIENTIST
    rename
      introduce as express
    redefine
      express
    end

create
  make_with_pet

feature {NONE} -- Initialization

  make_with_pet (a_name: STRING; a_pet: PET)
      -- Initialization for 'Current'.
    require
      name_exists: a_name /= Void and then not a_name.is_empty
      pet_exists: a_pet /= Void
    do
      make (a_name)
      pet := a_pet
    ensure
      name_set: name = a_name
      pet_set: pet = a_pet
    end

feature -- Access

  pet: PET
      -- Current biologist's pet.
```

```
feature -- Basic operations

  express
      -- Print info about self.
    do
      Precursor
      print ("I am a biologist. ")
      print ("I have a pet. Its name is " + pet.name + ".")
    end
end
```

```
class
  PET

create
  make

feature {NONE} -- Initialization

  make (pet_name: STRING)
      -- Initialization for `Current'.
    require
      pet_name_exists: pet_name /= Void and then not pet_name.is_empty
    do
      name := pet_name
    ensure
      pet_name_set: name = pet_name
    end

feature -- Access

  name: STRING
      -- Current pet's name.

feature -- Basic operations

  introduce
      -- Print info about self.
    do
      io.put_new_line
      print ("My name is " + name + " and I tend to be afraid.")
    end
end
```

Indicate, for each of the code fragments below, if it compiles by checking the corresponding box. If the code fragment does not compile, explain why this is the case and clearly mark the line that does not compile. If the code fragment compiles, specify the text that is printed to the console when the code fragment is executed.

Given the following variable declarations:

```
a_scientist: SCIENTIST
a_computer_scientist: COMPUTER_SCIENTIST
a_biologist: BIOLOGIST
```

Example 1:

```
(create {PET}.make ("Bob")).introduce
```

Does the code compile? ☒ Yes ☐ No
Output/error description My name is Bob and I tend to be afraid.

Example 2:

```
Bob.introduce
```

Does the code compile? ☐ Yes ☒ No
Output/error description The code does not compile, because "Bob" is an unknown (not declared) identifier.

Task 1

```
create a_scientist.make ("Theo")
a_scientist.introduce
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 2

```
create a_computer_scientist.make ("Heidi")
a_computer_scientist.introduce
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 3

```
a_scientist := create {COMPUTER_SCIENTIST}.make ("Helen")
a_scientist.introduce
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 4

```
a_scientist := create {COMPUTER_SCIENTIST}.make ("Hal")
a_computer_scientist := a_scientist
a_computer_scientist.introduce
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 5

```
create a_biologist.make_with_pet ("Reto", create {PET}.make ("Toby"))
a_biologist.express
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 6

```
create a_biologist.make_with_pet ("Kandra", create {PET}.make ("Tom"))
a_computer_scientist := a_biologist
a_computer_scientist.introduce
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 7

```
a_biologist := create {BIOLOGIST}.make_with_pet ("Elmo", create {PET}.make ("
    Hex"))
a_scientist := a_biologist
a_scientist.pet.introduce
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

3 Specifying Software through Contracts (12 points)

The city *Lake* is a home to many libraries. Each of these libraries owns books and provides a book lending service to the public. Each book belongs to one library. The number of books a reader can borrow from a library is not limited, and a book can be lent to only one reader at a time.

Listings 1 through 3 show code snippets of the three classes *LIBRARY*, *READER*, and *BOOK* from the software system used to manage these libraries. Your task is to add contracts to the snippets, so that the informal description above and in the feature comments is reflected in the class interface.

Please note:

- You should read through all the feature comments first.
- The Eiffel compiler ensures that entities of type **attached** *T* are never **Void**. For example the attribute *books*: **attached** *ARRAYED_SET* [**attached** *BOOK*] can never receive **Void** or contain **Void** elements, and therefore contracts like *books* /= **Void** are unnecessary, and should not be written for such entities.
- That the explicit use of **attached** and **detachable** and the corresponding rules apply to this problem only. (Also, in standard Eiffel, types are **attached** by default, so **attached** is normally not specified. It is included in this example for clarity.)
- You need to write **True** at places where you think no explicit contract is needed: leaving a precondition, postcondition, or class invariant empty gives you 0 point for that section.
- The number of dotted lines is not indicative of the number of missing contract clauses.
- The following features from class *ARRAYED_SET* may be useful:

```
class ARRAYED_SET [G]

feature
    has (v: G): BOOLEAN
            -- Does current include 'v'?

    is_subset (other: like Current): BOOLEAN
            -- Is current set a subset of 'other'?

    -- Other features omitted.
end
```

Listing 1: Class *LIBRARY*

```
deferred class
    LIBRARY

feature -- Book

    books: attached ARRAYED_SET [attached BOOK]
            -- Set of books the library owns.

    has_book (a_book: attached BOOK): BOOLEAN
            -- Does the library own 'a_book'?
        require

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

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

            ..........................................................................
        deferred
        ensure

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

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

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

    acquire (a_book: attached BOOK)
            -- Add 'a_book' to the library.
        require

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

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

            ..........................................................................
        deferred
        ensure

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

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

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

    borrower (a_book: attached BOOK): detachable READER
            -- Reader who has borrowed but not yet returned 'a_book'.
            -- Return Void if none.
        require

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

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

```
            .........................................................................
        deferred
        ensure

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

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

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

feature -- Operation

    lend (a_book: attached BOOK; a_reader: attached READER)
            -- Lend 'a_book' to 'a_reader'.
            -- Applicable only to books owned by 'Current'.
        require

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

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

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

            .........................................................................
        deferred
        ensure

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

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

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

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

    recover (a_book: attached BOOK)
            -- Recover 'a_book' from its borrower.
            -- Applicable only to books owned by 'Current'.
            -- This is the reverse operation of 'lend'.
        require

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

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

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

            .........................................................................
        deferred
        ensure

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

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

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

-- Other features omitted.
invariant
    ..................................................................
    ..................................................................
    ..................................................................
    ..................................................................
end
```

Listing 2: Class *READER*

```
deferred class
    READER

feature -- Access

    borrowed_books (a_lib: attached LIBRARY): attached ARRAYED_SET [attached
        BOOK]
            -- Set of books borrowed from 'a_lib'.
            -- Return empty set if 'Current' has not currently borrowed
            -- any book from 'a_lib'.
        require
            ..................................................................
            ..................................................................
            ..................................................................
        deferred
        ensure
            ..................................................................
            ..................................................................
            ..................................................................
        end

-- Other features omitted.

invariant
    ..................................................................
    ..................................................................
    ..................................................................
end
```

Listing 3: Class *BOOK*

```
deferred class
    BOOK

feature -- Access

    library: detachable LIBRARY assign set_library
            -- Library to which the book belongs. Immutable once set.

feature{LIBRARY} -- Set

    set_library (a_lib: attached LIBRARY)
            -- Set 'library' to 'a_lib'.
        require

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

            ..........................................................................
        deferred
        ensure

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

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

-- Other features omitted.

invariant

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

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

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

4 Recursion (13 points)

Task 1

The function *n_th_element* (see below) should implement a recursive algorithm that, given a list *a*, computes the n-th element of a sorted list (in ascending order) that contains the same elements as the list *a*. Note that list *a* does not need to be sorted. See the example in task 2 to get an idea of what the correct output of function *n_th_element* should look like. *Complete the implementation by filling in the missing expressions. Note that the expected implementation uses recursion.*

```
n_th_element (a: ARRAYED_LIST [INTEGER]; n: INTEGER): INTEGER
    require
        a /= Void and 1 <= n and n <= a.count
    local
        i, element, pivot: INTEGER
        not_greater, greater: ARRAYED_LIST [INTEGER]
    do
        print ("n = " + n.out + "%N")
        pivot := a [n]

        -- Split up the elements of list 'a', such that 'not_greater' contains all elements
        -- that are less than or equal to 'pivot' (except for the pivot itself),
        -- and 'greater' contains all elements that are greater than 'pivot'.
        create not_greater.make (0)
        create greater.make (0)
        from
            i := 1
        until
            a.count < i
        loop
            element := a [i]
            if element <= pivot and i /= n then
                not_greater.extend (element)
            elseif pivot < element then
                greater.extend (element)
            end
            i := i + 1
        end

        if (.................................................................) then

            Result := (...........................................................)

        elseif (.............................................................) then

            Result := (...........................................................

                       ...........................................................)

        elseif (.............................................................) then

            Result := (...........................................................

                       ...........................................................)
        end
    end
```

Task 2

In the following code snippets, function *n_th_element* is called with different inputs. *Write down the output that is printed to the console for each snippet once function n_th_element has been properly implemented.* Note that the function *n_th_element* prints out the argument *n* in each call.
Assume that variable *a* was declared as follows:

```
local
    a: ARRAYED_LIST [INTEGER]
```

Example

```
create a.make (0) -- Create an empty list.
a.extend (1)
a.extend (2)
a.extend (−2)
print ("result = " + n_th_element (a, 1).out)
```

Output:

```
n = 1
n = 1
result = -2
```

Snippet 1

```
create a.make (0)
a.extend (0)
print ("result = " + n_th_element (a, 1).out)
```

Output:

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

Snippet 2

```
create a.make (0)
a.extend (5)
a.extend (1)
a.extend (3)
a.extend (2)
a.extend (1)
print ("result = " + n_th_element (a, 5).out)
```

Output:

....

....

....

....

....

....

5 Data Structures (16 points)

In this task you are going to implement several operations for a generic class *SET* [*G*].

A set is a collection of distinct objects. Every element of a set must be unique; no two members may be identical. All set operations preserve this property. The order in which the elements of a set are listed is irrelevant (unlike for a sequence or tuple). Therefore the two sets $\{5, 10, 12\}$ and $\{10, 12, 5\}$ are identical.

There are several fundamental operations for constructing new sets from given sets.

- Union: The union of A and B, denoted by $A \cup B$, is the set of all elements that are members of either A or B.
- Intersection: The intersection of A and B, denoted by $A \cap B$, is the set of all elements that are members of both A and B.
- Relative complement of B in A (also called the set-theoretic difference of A and B), denoted by $A \backslash B$ (or $A - B$), is the set of all elements that are members of A but not members of B.

The Jaccard index (or coefficient) measures similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets (see Figure 2). If both sets are empty the Jaccard coefficient is defined as 1.0.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Figure 2: Jaccard index definition for non-empty sets A and B.

Your task is to fill in the gaps of class *SET* [*G*] below. Please note:

- Your code should satisfy the contracts and provide new contracts where necessary.
- The set should never contain **Void** elements.
- The number of dotted lines does not indicate the number of missing contract clauses or code instructions.
- The implementation of class *SET* [*G*] is based on an arrayed list. The arrayed list is set up to use object comparison, so features like *has* and *prune* use object equality instead of reference equality when comparing elements from the set. The following features of class *ARRAYED_LIST* may be useful:

```
class ARRAYED_LIST [G]

feature
  has (v: G): BOOLEAN
    -- Does current include 'v'?

  start
    -- Move cursor to first position if any.

  extend (v: G)
    -- Add 'v' to the end.

  prune (v: G)
    -- Remove first occurrence of 'v', if any, after cursor position.
    -- Move cursor to right neighbor.

-- Other features are omitted.
end
```

```
class
  SET [G]

create
  make_empty

feature {NONE} -- Initialization

  make_empty
      -- Create empty Current.
    do
      create content.make (0)
      content.compare_objects
    ensure
      empty_content: content.is_empty
    end

feature -- Access

  count: INTEGER
      -- Cardinality of the current set.
    do
      Result := content.count
    end

  is_empty: BOOLEAN
      -- Is current set empty?
    do

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

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

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

    end

  has (v: G): BOOLEAN
      -- Does current set contain 'v'?
    require

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

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

    do

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

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

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

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

    end
```

```
add (v: G)
    -- Add 'v' to the current set.
  require
    .............................................................................
    .............................................................................
  do
    .............................................................................
    .............................................................................
    .............................................................................
    .............................................................................
  ensure
    .............................................................................
    .............................................................................
    .............................................................................
  end

remove (v: G)
    -- Remove 'v' from the current set.
  require
    .............................................................................
    .............................................................................
  do
    .............................................................................
    .............................................................................
    .............................................................................
    .............................................................................
  ensure
    .............................................................................
    .............................................................................
    .............................................................................
  end
```

```
  duplicate: like Current
      -- Deep copy of Current.
    do

      create Result.make_empty
      across content as c
      loop
        Result.add (c.item)
      end
    ensure

      same_size: Result.count = count
      same_content: across content as c all Result.has (c.item) end
    end

feature -- Set operations.

  union (another: like Current): like Current
      -- Union product of the current set and 'another' set.
    require

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

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

    do

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

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

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

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

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

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

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

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

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

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

    ensure

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

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

    end
```

```
intersection (another: like Current): like Current
    -- Intersection product of the current set and 'another' set.
  require

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

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

  do

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

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

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

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

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

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

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

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

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

  ensure

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

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

  end

difference (another: like Current): like Current
    -- Set-theoretic difference of the current set and 'another' set.
  require

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

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

  do

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

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

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

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

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

```
        ..............................................................................
        ..............................................................................
        ..............................................................................
        ..............................................................................
    ensure
        ..............................................................................
        ..............................................................................
    end

feature -- Set metrics.

  jaccard_index (another: like Current): REAL_64
      -- Jaccard similarity coefficient between current set and 'another' set.
    require
        ..............................................................................
        ..............................................................................
    do
        ..............................................................................
        ..............................................................................
        ..............................................................................
        ..............................................................................
        ..............................................................................
        ..............................................................................
        ..............................................................................
        ..............................................................................
        ..............................................................................
    ensure
        ..............................................................................
        ..............................................................................
    end
```

```
feature {NONE} -- Implementation

  content: ARRAYED_LIST[G]
      -- Items of the set.

invariant

  content_exists: content /= Void
  content_object_comparison: content.object_comparison
  non_negative_cardinality: count >= 0

end
```