ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Chair of Software Engineering
Bertrand Meyer, Manuel Oriol

Trusted Components

17 December 2007

Name, First name: .....

Stud.-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:

- Except for a dictionary and personal notes, you are not allowed to use any supplementary material.
- Please write your student number onto **each** 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.
- Manage your time carefully (take into account the number of points for each question).
- Please **immediately** tell the supervisors of the exam if you feel disturbed during the exam.
- The maximum duration of the examination is 1h45mn the minimum duration is 1h.

Good Luck!

Stud.-Number:.....

Question	Number of possible points	Points
1	27	
2	15	
3	18	
4	15	
Total	75	

Grade:.....

1 Axiomatic semantics (27 points)

1.1 (5 points) Write the partial correctness inference rule of axiomatic semantics for loops `from a until c loop b end'.

Hints:

- This is an inference rule whose purpose is to deduce {P} L {Q} where L is `from I until e loop B end'.
- The rule involves an assertion that may be called INV.

1.2 (1 point) Consider the following form of recursive function, with one (natural) integer argument and an integer result:

```
[1]
f (n: NATURAL): NATURAL
   do
      if n = 0 then
         Result := c
      else
         Result := g (n, f (n-1))
      end
   end
```

where c is a constant and g is a two-argument function (whose value is defined entirely in terms of its arguments, i.e. without use of any other entity of the program).

An example is

```
factorial (n: NATURAL): NATURAL
   do
      if n = 0 then
         Result := 1
      else
         Result := n * factorial (n-1)
      end
   end
```

What are c and g in this example?

1.3 (1 point) We consider the following loop equivalent for recursive definitions ofthe form [1] (you don't need to prove that this equivalence is correct).

[2]

```
f (n: INTEGER): NATURAL
  local
    i: INTEGER
  do
    from
      i := 0
      Result := c
    until
      i = n
    loop
      i := i + 1
      Result := g (i, Result)
    end
  end
```

Apply this transformation (literally, that is to say, purely by program transformation) to produce a recursion-free version of `factorial'. There is no need to prove anything about this transformation, just apply it as given by [2].

1.4 (10 points) From the rule for loops (question 1) and the recursion-loop equivalence ([2]), give an inference rule for proving the partial correctness of recursive functions of the form [1].

Hints:

- You need to apply the assignment axiom.
- The rule uses a notion of invariant.

1.5 (5 points) Using the rule from question 4, prove the correctness of the `factorial' function in its original form.

1.6 (5 points) What notion should be added to the above framework to yield a rule covering total correctness? (Only the name of the notion is required, no further justification or explanation. Hint: take advantage of the notion used to prove total correctness for loops.)

2 Component design and testing (15 points)

Analyze the class NETWORK_STREAM, and answer the following questions:

2.1 (9 points) For the routines *make*, *descriptor* and *next_character*, decide whether some designing principles for components are violated. If so, give the name of these principles and explain where and why they are violated.

2.2 (6 points) Suppose we are only working with HTTP protocol, for the feature *is_url_valid* , design test cases. Include what is the input and what is the expected output. You do not need to implement the feature *is_url_valid*.

```
class
   NETWORK_STREAM

create    make

feature{NONE} -- Initialization

   make (a_url: STRING; a_buffer_size: INTEGER)
         -- Initialize current network stream with URL `a_url' and with buffer size `a_buffer_size'.
      require
         a_url_attached: a_url /= Void
         a_buffer_size_positive: a_buffer_size > 0
         a_url_valid: is_url_valid (a_url)
      do
         set_url (a_url)
         set_buffer_size (a_buffer_size)
         -- Some other initialization, including initializing `buffer'.
      ensure
         url_set: url.is_equal (a_url)
         buffer_size_set: buffer_size = a_buffer_size
      end

feature    -- Access

   url: STRING
         --URL associated with Current stream

   buffer_size: INTEGER
         -- Size of buffer used to read data
```

```
descriptor: URL_DESCRIPTOR
        -- Descriptor for current stream, containing URL scheme information such as http, ftp.
      do
        if not is_descriptor_calculated then
          internal_descriptor := descriptor_from_url (url)
          is_descriptor_calculated := True
        end
        Result := internal_descriptor
      ensure
        result_attached: Result /= Void
      end

  position: INTEGER
        -- Position in the stream

feature -- Status report

  is_url_valid (a_url: STRING): BOOLEAN
        -- Is `a_url' valid?
      require
        a_url_attached: a_url /= Void
      do
        …
      ensure
        -- Result is True if and only if `a_rul' is of correct format.
      end

feature -- Stream IO

  next_character: CHARACTER
        -- Next character from the stream
      local
        l_char_size: INTEGER
      do
          -- Retrieve number of bytes used to represent a character.
        l_char_size := {PLATFORM}.character_bytes

          -- Read `buffer' for new data if necessary.
        if buffer.is_empty or else buffer.count < l_char_size then
          read_buffer
        end
          -- Load a character from `buffer'.
        Result := buffer.item_as_character

          -- Increase `position' from current stream.
        position := position + l_char_size
      ensure
        position_increased: position = old position + {PLATFORM}.character_bytes
      end
```

feature -- Setting

set_buffer_size (a_size: INTEGER)
-- Set `buffer_size' with `a_size'.
require
a_size_positive: a_size > 0
do
…
ensure
buffer_size_set: buffer_size = a_size
end

set_url (a_url: STRING)
-- Set `url' with `a_url'.
require
a_url_attached: a_url /= **Void**
a_url_valid: is_url_valid (a_url)
do
…
ensure
url_set: url /= **Void and then** *url.is_equal (a_url)*
end

feature{*NONE*} -- Implementation

buffer: BUFFER
-- Buffer to store read data, used as cache.

internal_descriptor: **like** *descriptor*
-- Internal stream descriptor

is_descriptor_calculated: BOOLEAN
-- Has `descriptor' been calculated?

descriptor_from_url (a_url: STRING): **like** *descriptor*
-- Stream descriptor calculated from `a_url'
require
a_url_attached: a_url /= **Void**
a_url_valid: is_url_valid (a_url)
do
…
ensure
result_attached: **Result /= Void**
end

```
read_buffer
        -- Fill `buffer' up to `buffer_size' bytes.
        -- Assume that current stream is unbounded, so we won't hit the end of the stream.
    do
        …
    ensure
        buffer_refilled: buffer.count = buffer_size
    end

feature
  -- Rest of the class
end
```

3 Program analysis (18 points)

3.1 (3 points) What kind of analysis do you need to know if a variable may be used in the following of the program before it is overwritten?

3.2 (10 points) Make control-flow graph of the following program and apply the analysis to it:

```
a := 1

b := a + 2

c := 10

if  b > 3  then

    c := 3 - c

else

    b := a

end

b := a + b

Result := b
```

3.3 (5 points) How would you do to also store where the variables were defined? Explain informally.

4 Abstract Interpretation (15 points)

Consider the following language:

$i \in [MIN\_INT, ..., MAX\_INT]$

e::= i | e1 * e2 | e1 + e2 | e1 - e2

The order of magnitude for MAX_INT and MIN_INT is around 10^{40} and -10^{40}.

4.1 (10 points) Create an abstraction to evaluate the value of expressions and ensure that the expressions value do not go over MAX_INT or below MIN_INT.

4.2 (5 points) Is the abstraction still valid if expressions are now defined as follows (// is the integer division):

e::= i | e1 * e2 | e1 + e2 | e1 - e2 | e1 // e2

If not, explain why and refine it.