



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Chair of Software Engineering
Bertrand Meyer, Manuel Oriol

Trusted Components

29. January 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	15	
2	20	
3	15	
4	15	
Total	60	

Grade:.....

1 Reuse and component quality (15 points)

1.1 (5 points) Assume you are asked to institute a “Reuse Policy” for your organization. What would be its major components?

1.2 (10 points) We saw the “active” style of API design for data structure components, such as lists with cursors.

- (3 points) Explain the basic idea of this design style.
- (3 points) Contrast it with another possible style, using a specific example (e.g. lists).
- (4 points) Discuss the pros and cons of the choice between the two styles.

2 Axiomatic Semantic (20 points)

We consider a simple language such as “Graal” used in class to discuss axiomatic semantics, and extend it with two notions: *clock counter* and *non-deterministic choice from integer intervals*. This means two new instructions, with the following possible concrete syntax:

- **clock** t
- **choose** t by e

In both cases t is an integer variable; in the second, e is an integer expression.

The **clock** instruction assigns to t the current value of the machine clock. The machine clock is positive, never has the same value twice, and is always increasing.

The **choose** instruction assigns to t an integer value in the interval $0..|e-1|$ (i.e. between 0 and the absolute value of $e-1$, inclusive). The implementation is free to use any value in that interval.

- 2.1. (10 points) Write axiomatic semantic definitions for these two instructions.
- 2.2. (10 points) Use your semantics to prove that the following loop always terminates:

```
from
  clock  $i$ 
until  $i = 0$  loop
  clock  $j$ 
  if  $i < j$  then
    choose  $i$  by  $i$ 
  end
end
```

3 Testing and Patterns (15 points)

3.1 Given the following Eiffel class, you are required to design test cases that include both input values of each routine and the target objects on which the routine is called. Illustrate the strategy you adopt in your test case design process and evaluate the quality of your test cases.

indexing

description: "STRINGS with copy-on-write semantics"
library: "Gobo Eiffel String Library"
copyright: "Copyright (c) 2005, Colin Adams and others"
license: "Eiffel Forum License v2 (see forum.txt)"
date: "\$Date: 2005/10/18 04:39:22 \$"
revision: "\$Revision: 1.3 \$"

class

ST_COPY_ON_WRITE_STRING

create

make

feature {NONE}

make (a_string: STRING) is

require

a_string_not_void: a_string /= Void

do

item := a_string

ensure

item_set: item = a_string

unsafe_to_edit: not changed

end

feature

item: STRING

safe_item: STRING is

do

clone_if_unchanged

Result := *item*

ensure

safe_to_edit: changed

same_as_item: Result /= Void and then Result = item

end

feature

put (c: CHARACTER; i: INTEGER) is

require

valid_index: item.valid_index (i)

do

clone_if_unchanged

item.put (c, i)

ensure

stable_count: item.count = old item.count

replaced: item.item (i) = c

safe_to_edit: changed

end

```

append_character (c: CHARACTER) is
  do
    clone_if_unchanged
    item.append_character (c)
  ensure
    new_count: item.count = old item.count + 1
    appended: item.item (item.count) = c
    safe_to_edit: changed
  end

insert_character (c: CHARACTER; i: INTEGER) is
  require
    valid_insertion_index: 1 <= i and i <= item.count + 1
  do
    clone_if_unchanged
    item.insert_character (c, i)
  ensure
    one_more_character: item.count = old item.count + 1
    inserted: item.item (i) = c
    safe_to_edit: changed
  end
feature {NONE}
  changed: BOOLEAN

clone_if_unchanged is
  do
    if not changed then
      item := string.cloned_string (item)
      changed := True
    end
  ensure
    same_item_if_previously_changed: old changed implies item = old item
    cloned_if_not_previously_changed: not old changed implies item /= old item
    safe_to_edit: changed
  end
invariant
  item_not_void: item /= Void
end

```

3.2 Compare the VISITOR pattern and the VISITOR component and illustrate the advantages and disadvantages of each approach.

4 Program Analysis (15 points)

4.1 Do the Control-Flow Graph (CFG) of the following routine:

```
calc (i: INTEGER): INTEGER is
  -- This function has no comment
local
  a, b, c, d, index: INTEGER
do
  a := i
  b := 0
  c := a + b
from
  Result := 1
  index := i
until
  index < 1
loop
  if c > 0 then
    c := a + b
    Result := 1
  else
    d := a + b
    Result := Result - d
  end
  index := index - 1
end
  c := a + b
  if c < 0 then
    Result := 0
  end
end
```

4.2 On the CFG that you created, write the results of the very busy expression analysis. Propose an optimization of the program based on the result of this analysis.

4.3 The program is obviously returning only 0 or 1. What kind of technique would you use to simplify the program? Why?