

Concepts of Programming Languages (INFOTECH)

Summer Semester 2008
Prof. Plödereder, Stefan Staiger/Steffen Keul

Assignment #2

Please submit your solution on Friday, July 4th 2008 in class or send it via email to keulsn@informatik.uni-stuttgart.de. (If you want to submit earlier, please do so.) Please submit either plain text or PDF, but not MS Word files.

1 Parameter Passing Mechanisms

The following source code is a program written in *pseudo-Ada* (i.e., one should not assume that the program uses the Ada parameter passing mechanism).

```
type Field is array (1..5) of Integer;
```

```
a : Field;
```

```
-- x = reversed field y
procedure p (x,y : Field; i : Integer) is
begin
  x (i) := y (6-i);
  if i < 5 then
    p (x,y,i+1);
  end if;
end p;
```

```
procedure main is
begin
  a := (1, 2, 3, 4, 5);
  p (a,a,1);
end main;
```

- a. What is the value of variable **a** after the call `p(a,a,1)` assuming the following parameter passing mechanisms for **x** and **y**? (assume that **i** is always passed by value)
- Call by value
 - Call by reference
 - Call by value/result (a.k.a. copy/restore)
 - Call by name

Draw the activation records on the stack to illustrate step-by-step what happens!

- b. Explain the meaning of *aliasing* (by using a small example). Which parameter passing mechanisms cause aliasing in the above example program?
- c. Given a choice for each parameter, which parameter passing mechanism should you use to get the expected behavior of the program?

2 Garbage collection

Consider the following program, written in an Ada-like programming language which uses garbage collection to manage memory.

```
-- some global variables
bigObject : access MyBigClass;
pointer : access MyBigClass;
otherPointer : access MyBigClass;

-- some procedure
procedure P is
begin
  bigObject := new MyBigClass;
  -- create a list of objects
  pointer := bigObject;
  for i := 1 .. 100000 loop
    pointer.all.next := new MyBigClass;
    pointer.all.myField := pointer.all.next;
    pointer := pointer.all.next;
  end loop;
  -- work with the objects
  doSomething (bigObject);
  -- free memory
  pointer := bigObject;
  while pointer /= null loop
    otherPointer := pointer;
    pointer := pointer.all.next;
    otherPointer.all.next := null;
  end loop;
end P;
```

The procedure `doSomething` does not modify the pointers.

- a. How does the garbage collector work?
- b. Assume the garbage collector tries to free memory
- before the call to `doSomething`,
 - at the end of `P`.

What memory can then be freed? If some memory cannot be freed, show the reason!

- c. What happens if `P` is called very often?
- d. How could we correct the code to avoid the problem?

3 Exceptions

Examine this program written in Ada 95:

```
01 with Ada.Text_IO; use Ada.Text_IO;
02 procedure Main is
03   E1, E2, E3 : exception;
04   procedure Q is
05     begin
06       raise E2;
07     exception
08       when E1 => Put_Line ("Q: E1"); raise E3;
09       when others => Put_Line ("Q: other exception"); raise E1;
10     end Q;
11   procedure P is
12     begin
13       begin
14         Q;
15       exception
16         when E1 => Put_Line ("P/1: E1"); Q; raise E2;
17       end;
18       begin
19         Q;
20       exception
21         when others => Put_Line ("P/2: other exception"); raise E1;
22       end;
23     exception
24       when E1 => Put_Line ("P/3: E1"); raise;
25       when E2 => Put_Line ("P/3: E2"); raise E3;
26       when E3 => Put_Line ("P/3: E3"); raise E3;
27     end P;
28 begin
29   P;
30 exception
31   when E1 => Put_Line ("Main: E1");
32   when E2 => Put_Line ("Main: E2");
33   when E3 => Put_Line ("Main: E3");
34   when others => Put_Line ("Main: other exception");
35 end Main;
```

Note: If an exception is raised (or re-raised) inside an exception handler, the same list of handlers is not checked again. Instead, the exception is propagated to the outside of the current block. It may then be caught by a handler of an outer block.

In what order are the statements in this program executed? What is the output of the program?