Programming Language Design
Assignment 2 2022

Torben Mogensen and Hans Hüttel

20th February 2022

This assignment is *individual*, so you are not allowed to discuss it with other students. All questions should be addressed to teachers and TAs. If you use material from the Internet or books, cite the sources. Plagiarism *will* be reported.

Assignment 1 counts 20% of the grade for the course, but you are required to get at least 33% of the possible score for every assignment, so you can not count on passing by the later assignments only. You are expected to use around 9 hours in total on this assignment, including the resubmission (if any). Note that this estimate assumes that you have solved the exercises suggested in slides/videos and participated in the plenary and TA sessions. If not, you should expect to use considerably more time for this assignment.

The deadline for the assignment is **Friday March 4 at 16:00 (4:00 PM)**. The individual exercises below are given percentages that provide a rough idea how much they count (and how much time you are expected to use on them). These percentages do *not* translate directly to a grade, but will give a rough idea of how much each exercise counts.

In normal circumstances, feedback will be given by your TA no later than March 14. Note that, even if the TAs find no errors in your submission, this is no guarantee that it is perfect, so we strongly recommend that you take time to improve your submission for resubmission regardless of the feedback you get. You can do so until **March 21 at 16:00 (4:00 PM)**. Note that resubmission is made as a separate mandatory assignment on Absalon. If you resubmit, the resubmission is used for grading, otherwise your first submission will be used for grading.

The assignments consists of four exercises. You should hand in a single PDF file with your answers and a zip-file with your code. Hand-in is through Absalon. Your submission must be written in English.

A2.1) (40%) **Garbage collection**

The file `GC.c` contains a C program that defines a data structure for LISP-like values (restricted to 30-bit unsigned integers, `()`, and pairs). All values are represented as 32-bit unsigned integers (using the type `uint32_t`). More details in `GC.c`.

The heap is an array, and allocation is simply by using the next element in the array with no reclamation of space. This is, obviously, not ideal.

The program also contains code that builds and prints data and produces garbage while doing so. The unmodified program will run long enough to print two trees, but will fail before the third tree is printed. Your modified program should run without error.

Hint: The first and third number of live nodes reported should be between 300 and 400, and the second number should be between 20 and 30.

a. Extend the program by adding garbage collection. You should not change the representation of values, but you can change how values are allocated, e.g., by using a freelist. The garbage collector should report the number of live nodes after garbage collection. Garbage collection is explicitly invoked by the statement

```
root = gc(root);
```

where *root* is a variable holding the index into the heap-array of the single root (so everything not reachable from this is considered garbage). Note that garbage collection may (or may not) change the location of the root, so a new root is returned.

You can choose to implement mark-sweep or copying collection.

Your report should detail the changes (if any) you make to value representation and show the result of running your GC program. Include your complete program in a zip-file, and mark the places you have modified or added code.

A2.2) (25%) **Parameter passing mechanisms**

In an Algol-like language, we can write

```
type matrix = array[1..n,1..n] of real;
...
procedure transpose(a: matrix; b: matrix);
var i,j : 1..n;
begin
   for i:= 1 to n do
      for j := 1 to n do
         a[i,j] := b[j,i]
end
```

a) Suppose we use value-result parameters for a and b and that q is an array of type matrix that corresponds to the matrix

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

What will happen, if we call transpose(q,q) and the parameters are written back from left to right? What will happen in the general case?

b) And what would happen if we use reference parameters?

c) Many programmers use a reference parameter instead of a value parameter in order to avoid unnecessary copying of large structures. Why can this be a dangerous practice?

A2.3) (10%) **Scope rules**

Some would explain static scope rules as follows: Any applied identifier occurrence is always bound to the *most recent* value given to a statically enclosing binding of that identifier.

Is it accurate to describe static scoping rules in this way? Use the following F# program to justify your answer – thinking of the b found in the body of mango can be useful. Start by finding the value of bongo.

```
let rec bizarre b q =
    let mango z = b
    in
        if b then (bizarre false mango)
             else (mango 17, q 17);;

let dummy = function x -> false

let bongo = bizarre true dummy;;
```

A2.4) (15%) **Program transformations**

Consider the F# program from the previous exercise.

```
let rec bizarre b q =
    let mango z = b
    in
        if b then (bizarre false mango)
             else (mango 17, q 17);;

let dummy = function x -> false

let bongo = bizarre true dummy;;
```

Use λ-lifting and defunctionalization to turn this program into one that uses first-order functions only.

A2.5) (10%) **Control structures**

In Algol 60, a **for** loop

for x := b1 **to** b2 step k **do** S

has a counter x that is incremented with a step value k in each traversal of the loop and has an initial value with value b_1 and a final value b_2. The loop is exited when the counter exceeds the final value, that is, when $x < b_2$ if $k < 0$ or when $x > b_2$ if $k > 0$. The step values and initial and final values of a **for** loop can be real numbers, so the counter has type **real**. What problems can arise here?