Principles of Programming Languages, 2012.07.06

Notes:
- Total available time: 2h
- You may use any written material you need
- You cannot use computers, phones or laptops during the exam

Exercise 1

a) Define an iterator for lists in Scheme, such that calling it returns an element. When there are no more elements, it returns symbol <<end>>.

E.g.
(define il (make-iter '(1 2)))
(il) returns *1*
(il) returns *2*
(il) returns *<<end>>*

b) Define an analogous iterator for vectors.
c) Define a new construct "for/in" which iterates on lists or vectors.

E.g.
(for x in '(1 2 3) (display x)) shows *123*
(for x in '#(c a s a) (display x)(display ".")) shows *c.a.s.a.*

Exercise 2

Consider the operation *revmap* which reverses a list and then performs a *map*:
i.e. *revmap (*2) [1,2,3]* is *[6,4,1]*
a) Define a Prolog implementation of *revmap* traversing the list only once. (HINT: use as many parameters as needed)
b) Define a pure and strict Haskell implementation of *revmap*, possibly with linear time complexity (assuming that the mapped function has constant time complexity) and traversing the list only once.

Exercise 3

You are currently working on the Padi project, a graphical interface for entry-level users. The performance team has observed several major slowdowns in some applications using Padi. It seems that the problem is due to some memory leaks related to the usage of classes *ScrollBar* and *TextPane*.
In order to identify the problem, it is needed to track memory allocation and de- allocation of instances of those classes. You have to modify the C++ sources of Padi in order to:

- every time an instance of *ScrollBar* or *TextPane* is allocated on the heap, the address and the size of the allocate block is be printed on standard error
- every time a memory block holding an instance of *ScrollBar* or *TextPane* is freed, a message reporting the freed address is be printed on standard error
- design and API allowing to print the current status of the heap, that is the list of each memory block - with the associated size - holding instances of *ScrollBar* and *TextPane* on the heap

The same problem also arises with instances of class Button. Can your solution be exploited to track all memory allocation and de-allocation of instances of *Button*?

Solutions

Ex 1

a)
```
(define (iter-list lst)
  (let ((cur lst))
    (lambda ()
      (if (null? cur)
        '<<end>>
        (let ((v (car cur)))
          (set! cur (cdr cur))
          v)))))
```

b)
```
(define (iter-vector vec)
  (let ((cur 0)
        (top (vector-length vec)))
    (lambda ()
      (if (= cur top)
        '<<end>>
        (let ((v (vector-ref vec cur)))
          (set! cur (+ cur 1))
          v)))))
```

c)
```
(define (iter-for block data iter)
  (let ((iterandum (iter data)))
    (let loop ((x (iterandum)))
      (if (not (eq? x '<<end>>))
        (begin
          (block x)
          (loop (iterandum)))))))

(define (iter-dispatch data)
  (cond ((vector? data) iter-vector)
        ((list?   data) iter-list)
        (else (assertion-violation #f "Unknown type" data))))

(define-syntax for
  (syntax-rules (in)
               ((_ var in data expr ...)
                (iter-for (lambda (var) expr ...)
                          data (iter-dispatch data)))))
```

Ex 2
a)
```
revmap(F,[X|Y],Z,W) :- call(F, X, X1), revmap(F,Y,[X1|Z],W).
revmap(_,[],X,X).
```

b)
```
revmap f = foldl' (\x -> \y -> (f y) : x) []
```

Ex 3

```
#include <iostream>
#include <map>
#include <cstdlib>

namespace plp {

class Trackable {
public:
  static void *operator new(size_t size) {
    void *addr = std::malloc(size);
    std::cerr << "Allocation: Addr = " << addr << " Size = " << size << std::endl;
    allocated[addr] = size;
    return addr;
  }
  static void operator delete(void *addr) {
    std::cerr << "Deallocate: Addr = " << addr << std::endl;
    allocated.erase(addr);
    std::free(addr);
  }
public:
  static void dump() {
    std::cerr << "### Memory Map ###" << std::endl;
    for(std::map<void *, size_t>::iterator i = allocated.begin(),
                                           e = allocated.end();
                                           i != e;
                                           ++i)
      std::cerr << "  Address = " << i->first << " Size = " << i->second
                << std::endl;
  }

private:
  static std::map<void *, size_t> allocated;
};

class ScrollBar : public Trackable { };
class TextPane : public Trackable { };
} // End namespace plp.

using namespace plp;

std::map<void *, size_t> plp::Trackable::allocated;

int main(int argc, char *argv[]) {
  ScrollBar *bar;
  TextPane *text;
  bar = new ScrollBar();
  text = new TextPane();
  Trackable::dump();
  delete bar;
  delete text;
  text = new TextPane();
  Trackable::dump();
  delete text;
}
```