Principles of Programming Languages, 2012.09.03

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, Scheme (11 pts)

Let us consider trees memorized in Scheme as hierarchical lists (e.g. numeric expressions like 
(+ 2 3 (- 4 5) (/ 3 2))
).

1. Define a short, purely functional version of procedure numnodes, that accepts a tree and returns its
   number of nodes (e.g. (numnodes '(+ 2 3 (/ 1 3) (- 2 2 4 -7))) should return 11).
2. Define a lower-level, efficient, purely iterative version of numnodes.
3. Comment the following code, giving meaningful names to capitalized elements (i.e. $H1$ $V1$ ...). Also, please show a meaningful example usage.

   (define (H1 H2)
     (call/cc (lambda (V1)
       (for-each
         (lambda (x)
           (call/cc (lambda (V2)
                         (V1 (cons x V2))
                      )))
       H2)
     'V3))

Exercise 2, Haskell + Prolog (9 + 6 pts)

1. Define a datatype $Exp$ to represent generic expressions containing symbols and numbers e.g.
   $b(b(3,4,5),node(d,e))$.
2. Declare $Exp$ as an instance of Show, such that we can obtain representation exactly like
   "b(b(3,4,5),node(d,e))" (i.e. deriving Show is considered unacceptable)
3. Define a function, called $subst$, that accepts an expression $e$ and two atoms, $x$ and $y$, and returns a new
   expression $e'$ where every instance of $x$ is replaced by $y$.
4. Define a simplified version of $subst$ in Prolog, considering that expressions are at most binary (e.g.
   $a(1,b(a,2))$ is acceptable, while $b(b(3,4,5),node(d,e))$ is not.

Exercise 3, C++ (6 pts)

Alice is a young programmer. She is starting learning C++. Figure X (below) is one of the first programs she
has written, a map of interconnections among some cities. Inner classes
ConstDerefIterator and City are reported in Figure Y and in Figure Z respectively.
She is very proud of his work. Now she wants to print the city map on screen. The output must looks like the
following:

[Milano] Como Lecco Pavia
[Como] Milano Lecco
[Lecco] Como Milano
[Pavia] Milano

Each city is printed on a different line. The city name is reported at the start of the line, surrounded with
square brackets. It is followed by a list of neighboring cities.
Alice talks about her project of printing the city map with Bob, who introduces Alice to some advanced C++ concepts. In particular, Bob tells Alice to try writing a generic function printGraph that exploits traits to print a graph on standard output. Unfortunately, Alice is not so skilled, so she asks you whether you can help her in writing the generic code.

You are required to:

1. Write a generic function printGraph, that taken a generic graph prints it on standard output according to Alice's format. It must obtain graph-specific information (e.g. the root of the graph, nodes, links, ...) through a trait called GraphTraits.

2. Specialize the GraphTraits class in order to allow printing Alice's city map through the generic function printGraph.

```
Figure X:

class Map {
public:
    class ConstDerefIterator { ... };
    class City { ... };

public:
    typedef ConstDerefIterator const_node_iterator;
    typedef ConstDerefIterator const_child_iterator;

public:
    const_node_iterator begin() const {
        return const_node_iterator(cities.begin());
    }

    const_node_iterator end() const {
        return const_node_iterator(cities.end());
    }

public:
    Map() { }
    Map(const Map &that); // Do not implement.
    const Map &operator=(const Map &that); // Do not implement.
    ~Map() {
        for(std::vector<City *>::iterator i = cities.begin(),
            e = cities.end();
            i != e;
            ++i)
            delete *i;
    }

public:
    City &add(const char *name) {
        cities.push_back(new City(name));
        return *cities.back();
    }

private:
    std::vector<City*> cities;
};
```
class ConstDerefIterator {
public:
    ConstDerefIterator(const ConstDerefIterator &that) :
       cur(that.cur) { }

    ConstDerefIterator &
    operator=(const ConstDerefIterator &that) {
        if(this != &that)
           cur = that.cur;
        return *this;
    }

private:
    ConstDerefIterator(std::vector<City *>::const_iterator cur) :
       cur(cur) { }

public:
    bool operator==(const ConstDerefIterator &that) const {
        return cur == that.cur;
    }

    bool operator!=(const ConstDerefIterator &that) const {
        return cur != that.cur;
    }

    const City &operator*() const { return **cur; }

    ConstDerefIterator &operator++() {
       cur++; return *this;
    }

    ConstDerefIterator operator++(int ign) {
        ConstDerefIterator ret = *this; cur++; return *this;
    }

private:
    std::vector<City *>::const_iterator cur;
};

Figure Z:

class City {
public:
    typedef ConstDerefIterator const_child_iterator;

public:
    const_child_iterator begin() const {
        return const_child_iterator(neigh.begin());
    }

    const_child_iterator end() const {
        return const_child_iterator(neigh.end());
    }

private:
City(const char *name) : name(name) { }

City(const City &that); // Do not implement.
const City &operator=(const City &that); // Do not implement.

public:
    City &addNext(City &next) {
        std::vector<City *> &nextNeight = next.neigh;

        neigh.push_back(&next);
        nextNeight.push_back(this);

        return *this;
    }

public:
    const std::string &getName() const {
        return name;
    }

private:
    std::string name;
    std::vector<City *> neigh;

    friend class Map;
};
Solutions

Ex 1.1
(define (numnodes f)
  (if (not (list? f)) 1
      (+ 1 (apply +
            (map numnodes (cdr f))))))

Ex 1.2
(define (numnodesns f)
  (define stack0 (list f)) ; a heap-located stack representation
  (let loop ((stack (cdr stack0))
    (res 1)
    (curr (car stack0)))
    (if (null? stack)
      res
      (loop (cdr stack)
        (+ 1 res)
        (car stack))))

Ex 1.3
solution: an iterator that returns a pair (value . continuation)
H1: iterator
H2: lst
V1: exit
V2: yield/continuation
V3: the-end

(let ((a (H1 '(1 2 3))))
  (if (not (eq? a 'V3))
    (begin
      (display (car a))(newline)
      (loop ((cdr a))))))

Ex 2.1
instance Show Atom where
  show (N a) = show a
  show (S a) = filter (\x -> x /= "]") $ show a

Ex 2.2
instance Show Exp where
  show (A a) = show a
  show (E a) = filter (\x -> x /= "]") $ show a
instance Show Exp where
  show (A x) = show x
  show (E x (y:ys)) = show x ++ "(" ++
    show y ++
    concatMap (\t -> "," ++ show t) ys ++ ")"

Ex 2.3
subst :: Exp -> Atom -> Atom -> Exp
subst (A t) x y = if x == t then (A y) else (A t)
subst (E t es) x y = (E (if x == t then y else t) es')
  where es' = map (\g -> subst g x y) es

Ex 2.4
subst(X,X,Y,Y) :- !.
subst(E,X,Y,E) :- atomic(E), !.
subst(E,X,Y,E1) :-
  E =.. [X,L,R], !,
  subst(L,X,Y,L1),
  subst(R,X,Y,R1),
  E1 =.. [Y,L1,R1].
subst(E,X,Y,E1) :-
  E =.. [H,L,R], !,
  subst(L,X,Y,L1),
  subst(R,X,Y,R1),
  E1 =.. [H,L1,R1].

Ex 3
template <typename Ty>
struct GraphTraits;

template <typename Ty>
void printGraph(const Ty &graph) {
  typedef typename GraphTraits<Ty>::NodeIterator node_iterator;
  typedef typename GraphTraits<Ty>::ChildIterator child_iterator;
  for(node_iterator i = GraphTraits<Ty>::node_begin(graph),
      e = GraphTraits<Ty>::node_end(graph);
      i != e;
    ++i) {
    std::cout << "[" << *i << "]
    for(child_iterator j = GraphTraits<Ty>::child_begin(*i),
        f = GraphTraits<Ty>::child_end(*i);
        j != f;
        ++j)
        std::cout << " " << *j;
    std::cout << std::endl;
template <>
struct GraphTraits<Map> {
  typedef Map::const_node_iterator NodeIterator;
  typedef Map::const_child_iterator ChildIterator;

  static NodeIterator node_begin(const Map &map) {
    return map.begin();
  }

  static NodeIterator node_end(const Map &map) {
    return map.end();
  }

  static ChildIterator child_begin(const Map::City &city) {
    return city.begin();
  }

  static ChildIterator child_end(const Map::City &city) {
    return city.end();
  }
};