Principles of Programming Languages

2012.09.24

Notes

- Total available time: 2h 30'.
- You may use any written material you need.
- You cannot use computers, phones or laptops during the exam

1 Haskell (10 points)

1. Please define a procedure `getCSV` that is used, in the Monad IO, to get a field in a Comma Separated Values textfile (assume as separators both the , and ; characters).

2. Define the `unless` procedure in the IO Monad. It takes two arguments, a condition c and a body b, and repeats b unless c becomes true (in a sense, it is a "dual" of `while`).

3. Alfio is experimenting with Haskell, and wants to define a very simple object-oriented-like system, where an object is just a container of a datum, and a list of "methods", i.e. functions having signature $a \to a \to a$, where a is the type of the datum.

   This is an example object defined by Alfio:

```
myob = Obj (5, [("add", \self -> \x -> self+x),
                ("sqr", \self -> \x -> self*self)]) -- x is ignored
```

   Please, help Alfio to define both the needed datatype, and the `call` function for invoking methods. E.g. `call myob "add" -2` should return 3.

2 Prolog (8 points)

1. Please, define a Prolog procedure to obtain from a list the last but one element. If the number of elements in the list is less than 2, it should fail.

2. Define a nested version of the previous procedure: when the last but one element of the list is another list, this new procedure should look to the last but one element, until it finds an atomic value. E.g.

```
?- nestedlastb1([1,2,[3,3,[5,2,2],6],4],X).
X = 2.
```

3 Scheme (7 points)

Scheme does not natively support matrices: there are just vectors. Of course, we can define a matrix as a vector of vectors, each having the same size.

1. Please define `make-matrix`, with three parameters: r, c, and $fill$. This procedure returns a matrix having r rows and c columns, with every cell initialized with the value $fill$.

   Hint: the standard library offers a procedure `make-vector` *with two arguments: the first is the requested vector size, while the second is the content we want to initialize the vector with.*

2. Define also the setter and the accessor, called `matrix-set!` and `matrix-ref` respectively, with the natural parameters.

4 C++ (7 points)

You are following a class about concurrent programming. Your teacher has introduced pthread locks as the basic synchronization primitives for UNIX-like systems. All examples of the course are given using the C language, but your friend Charlie wants to do some concurrent C++. You have to help him defining a Lock class on top of a pthread_mutex_t. The pthread library provides the following functions for handling locks:

initialization **int** pthread_mutex_init(pthread_mutex_t *mutex, ...)

destruction **int** pthread_mutex_destroy(pthread_mutex_t *mutex)

lock **int** pthread_mutex_lock(pthread_mutex_t *mutex)

unlock **int** pthread_mutex_unlock(pthread_mutex_t *mutex)

Please note that pthread_mutex_init is a function with two arguments. The second argument – not reported – is used to initialize the mutex with custom properties. Passing NULL simply initialize the mutex with default properties.

Charlie is very satisfied of the Lock class, but he tell you that it is not very usable. The problem is that Charlie writes a lot of functions that acquire a lock, do something, and just before returning to the caller, release the lock. You are required to exploit the RAII C++ idiom to provide a more usable interface for handling locks in this setting.

In order to show Charlie the power of C++, you have to define a simple template class implementing a synchronized queue. It must define two member functions, pop and push. Each of them must be executed atomically. Elements are stored in a normal container – e.g. a std::deque – which is selected by means of a template parameter. You can assume that this template parameter provides push_back and pop_front member functions to insert and remove elements.

Solutions

Haskell

getCSV can be seen as a simple variant of getLine in the Prelude:

```
getCSV :: IO String
getCSV = do { c <- getChar;
        if c == ',' or c == ';' then return ""
        else do { l <- getCSV;
                return (c:l) }} -- NB: string == list of chars

unless test action = do
  val <- test
  if not val then do { action ; unless test action}
         else return ()

data SimpleObj a = Obj (a, [ (String , a -> a -> a) ])

find x [] = Nothing
find x ((a,b):xs) | x == a = Just b
find x (t:xs) = find x xs

call (Obj (self, methods)) name val =
    let Just met = find name methods in
    met self val
```

Prolog

```
lastb1([X,Y], X) :- !.
lastb1([X|L], Y) :- lastb1(L,Y).

nestedlastb1(L, X) :- lastb1(L,L1), atomic(L1), !, X = L1.
nestedlastb1(L, X) :- lastb1(L,L1), nestedlastb1(L1,X).
```

Scheme

```
(define (make-matrix max-rows max-cols fill)
  (let ((vec (make-vector max-rows #f)))
    (let loop ((x 0))
      (if (< x max-rows)
          (begin
            (vector-set! vec x (make-vector max-cols fill))
            (loop (+ x 1)))
          vec))))

(define-syntax matrix-set!
  (syntax-rules ()
    ((_ the-array row col val)
     (vector-set! (vector-ref the-array (- row 1)) (- col 1) val))))

(define-syntax matrix-ref
  (syntax-rules ()
    ((_ the-array row col)
```

```
    (vector-ref (vector-ref the-array (- row 1)) (- col 1)))))
```

C++

```
#include <deque>
#include <iostream>
#include <cstdlib>
#include <pthread.h>

namespace plp {

class Lock {
public:
  Lock() {
    pthread_mutex_init(&lock, NULL);
  }

  ~Lock() {
    pthread_mutex_destroy(&lock);
  }

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

public:
  void acquire() { pthread_mutex_lock(&lock); }
  void release() { pthread_mutex_unlock(&lock); }

private:
  pthread_mutex_t lock;
};

class ScopedLock {
public:
  ScopedLock(Lock &lock) : lock(lock) {
    lock.acquire();
  }

  ~ScopedLock() {
    lock.release();
  }

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

private:
  Lock &lock;
};

template <typename Ty, typename ContainerTy = std::deque<Ty> >
class SynchronizedQueue {
public:
  SynchronizedQueue() { }
```

```
  // Do not implement.
  SynchronizedQueue(const SynchronizedQueue &that);

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

public:
  void push(const Ty &elt) {
    ScopedLock lock(thisLock);

    els.push_back(elt);
  }

  void pop() {
    ScopedLock lock(thisLock);

    els.pop_front();
  }

public:
  void dump() const {
    std::cerr << *this << std::endl;
  }

public:
  friend std::ostream &operator<<(std::ostream &os,
                                  const SynchronizedQueue &queue) {
    SynchronizedQueue<Ty, ContainerTy> &noConst =
      const_cast<SynchronizedQueue<Ty, ContainerTy> &>(queue);
    ContainerTy &els = noConst.els;

    typedef typename ContainerTy::iterator iterator;

    ScopedLock lock(noConst.thisLock);

    os << "[ ";
    for(iterator i = els.begin(), e = els.end(); i != e; ++i)
      os << *i << " ";
    os << "]";

    return os;
  }

private:
  Lock thisLock;
  ContainerTy els;
};

} // End namespace plp.

using namespace plp;

int main(int argc, char *argv[]) {
```

```
  SynchronizedQueue<int> queue;

  queue.push(3);
  queue.push(5);
  queue.push(7);

  queue.dump();

  queue.pop();
  queue.pop();

  queue.dump();

  return EXIT_SUCCESS;
}
```