COM2001: Abstract Data Types

1. A software designer has been asked to create a new data type called *Library* for a Haskell-based library-loan system.[1] The following algebraic data types have already been defined; they represent the library's books, readers and loans. For example, the loan `(Book 24, Reader 5)` represents the fact that the book with ID number 24 is currently on loan to the reader with ID 5.

```
data Book   = Book { bookID :: Int }
data Reader = Reader { readerID :: Int }
type Loan   = (Book, Reader)
```

   The designer wants the data type *Library* to include functions that record

   - the library's books (no two books can have the same ID);
   - the library's readers (no two readers can have the same ID);
   - whether or not a book is currently on loan, and if so, to which reader. A book can only be issued to a reader if
     - the book is listed in the library's collection;
     - the reader is a registered user of the library;
     - the book is not already on loan.

   The required functions are:

Function	Required Behaviour
newLib	Create a new library with no books, readers or loans
getBooks	Given a library, return a list of all books in that library
getReaders	Given a library, return a list of all readers using that library
getLoans	Given a library, return a list of all current loans
addBook	Given a library and a book, add the book to the library and return the updated library
addReader	Given a library and a book, add the reader to the library and return the updated library
addLoan	Given a library, a book and a reader, issue the book to the reader and return the updated library
delBook	Given a library and a book, remove the book from the library's collection and return the updated library
delReader	Given a library and a reader, remove the reader from the library's list of registered users and return the updated library
delLoan	Given a library, a book and a reader, cancel the loan of the book to the reader, and return the updated library

   a) Which of the functions might potentially generate error conditions? Explain the circumstances under which each error would occur.

   b) Write down the syntax of the required ADT.

[1] NOTE: This specification is potentially ambiguous. You can resolve the ambiguities any way you like, but be sure to (a) say where you've done so, and (b) explain what you've done and why.

c) Write down the *Library* ADT's sorts, together with any specific values that need to be defined (for example, if one of the sorts is *Bool*, then required values of type *Bool* might be *true* and *false*). Explain what the sorts and values are used to represent.

d) Which of *Library*'s functions are constructors? How many rules are needed to describe *Library*'s semantics?

e) Write down the semantics of the required ADT, and explain what each rule means in plain English.

2. A programmer wants to implement a *deque* (double-ended queue). A deque consists of a sequence of values – you can insert values at either end of the deque, and (provided the deque isn't empty) you can query and remove values at either end. The programmer wants the ability to perform (at least) the following operations.

   - create : Takes no parameters, and returns the empty deque.
   - addFront, addBack : These add an entry to the relevant end of the deque and return the new deque.
   - empty : Tests whether a deque is or is not empty.
   - removeFront, removeBack : These remove an entry from the relevant end of the deque and return the resulting deque.
   - front : Returns the entry currently at the front of the deque.
   - back : Returns the entry currently at the back of the deque.

   a) Design an ADT called **Deque** that satisfies the programmer's requirements; remember to include the relevant sorts, syntax and semantics.

   b) Write down a Haskell implementation of your ADT.