Exercises 5 (Abstract Data Types)

5A The so-called 'millenium bug' was a major problem for the software industry in the late 1990s. In numerous application programs and databases, each date was represented by a string of six decimal digits: two digits for the day number, two digits for the month number, and two digits for the year number. For example, 31 December 1999 was represented by "311299", the leading digits "19" of the year number being implicit. Thus there was no means to represent dates after (or before) the 1900s.

In principle, the solution was simple: what? In practice, the problem was enormous: why?

5B Enhance the `Date` class in the course notes. Provide sufficient operations to allow application code to manipulate `Date` objects in all reasonable ways – but provide no more operations than necessary.

5C Consider the `String` contract in the course notes.

(a) Derive each operation's time complexity, assuming the array representation of strings.

(b) With the SLL representation, how is it possible to implement `concat` with time complexity $O(n)$, where n is the length of the first string, i.e., independent of the length of the second string?

5D A *text* is a sequence of characters subdivided into lines. One application of a text is the internal buffer of a text editor.

(a) Design a text ADT suitable to support a text editor. Assume the usual editing facilities: insertion and deletion of characters, cut, copy, and paste.

(b) Choose a representation for texts, and implement your ADT in Java.

5E Design an ADT to represent a time of day. What operations do you need to provide?

Implement your ADT in Java in at least two different ways. Which implementation is the easier?

5F Design an ADT to represent a university course. Assume that a course has a course code, a course title, at least one instructor, and one or more teaching assistants. What operations do you need to provide?

Implement your ADT as a Java class.

Exercises 6 (Stack ADTs)

6A Hand-test the bracket matching algorithm with the following phrases:

(a) `((4+8)*(3-2)`

(b) `{a,b,c} + {d,e,f}`

(c) `main(String[] args){print(arg[0]);}`

6B Assuming the array representation of stacks, show the contents of the array while each of the phrases in Exercise 6A is checked.

6C Repeat Exercise 6B assuming the SLL representation of stacks.

6D Consider the following additional requirements for a stack ADT: it should be possible to obtain a stack's current size; and it should be possible to access (but not remove) the element at any given depth *d* in a stack. (For example, $d = 1$ would access the topmost element in the stack.)

(a) Modify the stack contract accordingly.

(b) Modify the array implementation.

(c) Modify the SLL implementation.

6E In the array implementation of stacks, an ***overflow*** occurs when the stack size is about to exceed the array length.

(a) Show how to deal with an overflow by throwing an exception.

(b) Show how to deal with an overflow by substituting a longer array. Copy all the stack elements into a new and longer new array, and replace the existing array by that new array.

Exercises 7 (Queue ADTs)

7A In the cyclic-array implementation of queues, make the `add` method deal with an overflow by substituting a longer array. It should copy all the queue elements into a new and longer new array, and replace the existing array by that new array.

7B In the `ArrayQueue` class, show that one of the three instance variables `size`, `front`, and `rear` can be dropped. Which one? Modify the implementation accordingly.

7C Would you implement the queue ADT using a doubly-linked-list? Explain your answer.

7D (For those familiar with UNIX.) Outline how a UNIX pipe can be implemented by a queue.

7E A *keyboard driver* is a process (generally provided by the operating system) that allows the user to enter characters at any speed, without waiting for the application program to use these characters.

(a) Outline how the keyboard driver can communicate with the application program via a queue.

(b) Write an algorithm for the keyboard driver. It should 'echo' to the screen each graphic character (visible character or space) entered by the user. It should simply ignore any control character.

(c) Suppose now that the DELETE character is to cancel the last graphic character. A sequence of DELETEs is to cancel the same number of graphic characters. What changes are needed to the keyboard driver, and to the communication between the keyboard driver and the application program?

* **7F** A ***doubly-ended queue*** (or ***deque***) allows elements to be added or removed at both ends of the queue. In other words, it supports the following operations in addition to the usual queue operations: *d*`.addFirst(`*x*`)` adds *x* at the front of deque *d*; *d*`.removeLast()` removes the element at the rear of deque *d*; *d*`.getLast()` retrieves the element at the rear of deque *d*.

Write a deque contract in the form of a Java interface named `Deque`.

Write a linked implementation of a deque as a Java class `LinkedDeque`. Make sure that all deque operations have time complexity $O(1)$.

Exercises 8 (List and Iterator ADTs)

8A Compare the `Stack` and `Queue` ADTs with the `List` ADT.

(a) Show that the `Stack` ADT is a special case of the `List` ADT. Which list operations correspond to `push`, `pop`, and `peek`?

(b) Show that the `Queue` ADT is a special case of the `List` ADT. Which list operations correspond to `addLast`, `removeFirst`, and `getFirst`?

8B Using the `List` ADT, write the following method:

```
static List<Person> reorder (
                List<Person> persons) { … }
// Assume that the elements of persons are ordered by name.
// Return a similar list, ordered such that all children (aged
// under 18) come before all adults (aged 18 or over), but
// otherwise preserving the ordering by name.
```

Assume that each `Person` object has public instance variables `name` and `age`.

8C Use an iterator to extend the simple text editor in the course notes with the following methods: `findFirst(`*s*`)` selects the first line in the text that has a substring *s*; `findNext()` selects the next line in the text that has the substring *s* that was most recently supplied as an argument to `findFirst`. If there are no further occurrences of *s*, the selected line is unchanged. To search the entire text, call `findFirst` once, then call `findNext` as often as necessary.

8D Suppose that the following operations are to be added to the `List` interface. The operation *l*`.contains(`*x*`)` returns true if and only if object *x* is an element of list *l*. The operation *l*`.indexOf(`*x*`)` returns the index of object *x* in *l* if it is an element, or –1 otherwise.

(a) Modify the `ArrayList` class to implement these operations.

(b) Modify the `LinkedList` class to implement these operations.

8E Suppose that the following operation is to be added to the `List` interface. The operation *l*`.subList(`*i*`, `*j*`)` is to return a new list that contains all of the elements in *l* with indices *i* through *j*–1.

(a) Modify the `ArrayList` class to implement this operation.

(b) Modify the `LinkedList` class to implement this operation.

8F Modify the `ArrayList` implementation to deal with the possibility of an overflow. Copy all the list elements into a new and longer new array, and replace the existing array by that new array.

8G Suppose that `getFirst`, `getLast`, `addFirst`, `addLast`, `removeFirst`, and `removeLast` operations were added to the `List` interface.

(a) Modify the `ArrayList` class to implement these operations.

(b) Modify the `LinkedList` class to implement these operations.

Modify the data representations if you think fit.

8H The class `java.util.LinkedList` represents a list by a DLL, rather than an SLL. Justify this decision. Show how you would modify the `LinkedList` implementation to use a DLL.