

Informatics 1–Functional Programming
Mock Programming Exam, 22–26 November 2010

1. Note that **ALL QUESTIONS ARE COMPULSORY**
2. **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS.** Take note of this in allocating time to questions.

1. Suppose the playing cards in a standard deck are represented by characters in the following way: '2' through '9' plus '0' (zero) stand for the number cards, with '0' representing the 10, while the 'A', 'K', 'Q' and 'J' stand for the face cards, i.e. the ace, king, queen and jack, respectively. Let's call these the 'card characters'. The other characters, including the lowercase letters 'a', 'k', 'q', and 'j', and the digit '1', are not used to represent cards.

(a) Write a function `f :: String -> Bool` to test whether all card characters in a string represent face cards. For example:

```
f "ABCDE" = True      f "none here" = True      f "4 Aces" = False
f "01234" = False     f "" = True                f "1 Ace" = True
```

Your function can use *basic functions*, *list comprehension* and *library functions*, but not recursion. Credit may be given for indicating how you have tested your function.

[12 marks]

(b) Write a function `g :: String -> Bool` that behaves like `f`, this time using *basic functions* and *recursion*, but not library functions or list comprehension. Credit may be given for indicating how you have tested your function.

[12 marks]

(c) Write a function `h :: String -> Bool` that behaves like `f` using one or more of the following higher-order functions:

```
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
foldr :: (a -> b -> b) -> b -> [a] -> b
```

You can also use *basic functions*, but not other library functions, recursion or list comprehension. Credit may be given for indicating how you have tested your function.

[12 marks]

2. (a) Write a polymorphic function $\mathfrak{t} :: [a] \rightarrow [a]$ that duplicates every other item in a list. The result should contain the first item once, the second twice, the third once, the fourth twice, and so on. For example,

```
 $\mathfrak{t}$  "abcdefg" == "abbcdeffg"  
 $\mathfrak{t}$  [1,2,3,4] == [1,2,2,3,4,4]  
 $\mathfrak{t}$  []        == []
```

Your definition may use *basic functions*, *list comprehension*, and *library functions*, but not recursion. Credit may be given for indicating how you have tested your function. [16 marks]

- (b) Write a second function $\mathfrak{u} :: [a] \rightarrow [a]$ that behaves like \mathfrak{t} , this time using *basic functions* and *recursion*, but not list comprehension or library functions. Credit may be given for indicating how you have tested your function. [16 marks]

3. We introduce a data type to represent collections of characters.

```
data Chars = Range Char Char
           | Union Chars Chars
           | Complement Chars
```

For example,

```
Range 'a' 'z'
```

represents any lower case letter, and

```
Range 'A' 'Z'
```

represents any upper case letter, and

```
Union (Range 'a' 'z') (Range 'A' 'Z')
```

represents any letter, either lower or upper case, and

```
Complement (Union (Range 'a' 'z') (Range 'A' 'Z'))
```

represents any character that is not a letter.

(a) Write a function

```
showChars :: Chars -> String
```

that converts a collection of characters to a printable representation. Print a range of characters between square brackets separated by a dash, and a union between round brackets separated by a bar, and a complement by preceding it with a tilde. For example,

```
showChars (Range 'a' 'z') == "[a-z]"
showChars (Range 'A' 'Z') == "[A-Z]"
showChars (Union (Range 'a' 'z') (Range 'A' 'Z'))
  == "([a-z] | [A-Z])"
showChars (Complement (Union (Range 'a' 'z') (Range 'A' 'Z')))
  == "~([a-z] | [A-Z])"
```

Credit may be given for indicating how you have tested your function.

(You may assume that ranges contain only printable characters, and do not contain any of the characters "[] () - | ~".)

[16 marks]

(b) Write a function

```
inChars :: Chars -> Char -> Bool
```

that returns true if the given collection of characters contains the given character. For example,

```
inChars (Range 'a' 'z') 'd' == True
inChars (Range 'a' 'z') 'D' == False
inChars (Range 'A' 'Z') 'd' == False
inChars (Range 'A' 'Z') 'D' == True
inChars (Union (Range 'a' 'z') (Range 'A' 'Z')) 'd' == True
inChars (Union (Range 'a' 'z') (Range 'A' 'Z')) 'D' == True
inChars (Union (Range 'a' 'z') (Range 'A' 'Z')) '3' == False
inChars (Complement (Union (Range 'a' 'z')
                             (Range 'A' 'Z'))) 'd' == False
inChars (Complement (Union (Range 'a' 'z')
                             (Range 'A' 'Z'))) 'D' == False
inChars (Complement (Union (Range 'a' 'z')
                             (Range 'A' 'Z'))) '3' == True
```

Credit may be given for indicating how you have tested your function. [16 marks]