Swansea University
Department of Computer Science

U Berger
14 February 2011

CS_191 Functional Programming I

Programming Laboratory 2

Booleans, numbers, characters

In this Programming Laboratory session Exercises 1 and 3 are to be solved. Exercise 2 is voluntary. As always, you may work in pairs. The use of the text editor Emacs for writing your programs is compulsory. Please consult the course webpage for instructions of how to use Emacs and the course notes and the first Programming Labaratory sheet for instruction on how to create and use a Haskell script.

When your solutions are complete, please show them to a lab supervisor for assessment. You are expected to be able to explain your solutions and run the functions you defined with some test data.

Note that a complete definition of a Haskell function consists of the function's signature (that is, a declaration of the type of the function) *and* its defining equation.

Exercise 1 Define the logic gate `nand` which takes two Boolean values as inputs and returns a Boolean value as output, and which has the property that the output is `False` exactly when both inputs are `True`.

Give two definitions of `nand`: from predefined gates and by listing its truth table.

Exercise 2 (voluntary) Define `&&` using `nand` only. You are not allowed to use the constructors `True` or `False` or any other logic gate (except, of course in the definition of `nand`).

Hint: Define `not` first.

In order to avoid a clash of names with the predefined functions `&&` and `not` invent your own names for these functions, for example, `&&&` and `nott`.

See the course notes regarding the syntax of function names composed from special characters (such as, for example, `&`) and their use in prefix and infix notation.

Exercise 3 Define a function that tests whether a character is a digit, that is, one of the characters `'0'`, ..., `'9'`.

Recall from the lecture that the ASCII codes of characters denoting digits are arranged in a block of consecutive numbers. You can find out where this block begins and ends by computing the ASCII codes of the characters `'0'` and `'9'`. To this end evaluate the expressions `fromEnum '0'` and `fromEnum '9'`.