F28PL - Programming Languages

Question A (Basic SML)

Tommy Lamb

H00217505

The plain text file which accompanies this PDF contains a listing of all functions alongside test function calls, with some formatting to aid readability. It can be opened and executed in Poly/ML from the Terminal by navigating to the save location of the file, opening Poly/ML, and using the command : ' use "F28PL - Tommy Lamb - Question A - Code.txt"; '. This method will cause Poly/ML to ignore the document formatting and only display system output (not input). To maintain formatting, all text can be copied and pasted into a running instance of Poly/ML. For best results it should be copied/pasted in small blocks, around 1 question in size.

1. Multiply real `x` by integer `y` to give a `real`

```
- realIntMult 3.3 3;
>  9.9 : real
```

   - `fun realIntMult x y = x*(real y);`

     Simply converts the given integer `y` to type real before applying the in-built `real * real -> real` multiplication operator.

2. Check if string `s1` has more letters than string `s2`

```
- moreLetters "big" "small";
> false : bool
```

   - `fun moreLetters s1 s2 = (String.size s1) > (String.size s2);`

     Calls the in-built function `String.size` on each string, before comparing the length of each string and returning the boolean result of the comparison.

3. Check if character c represents a digit. Do not answer this by writing just `Char.isDigit`. Also, **do not hard-code** ASCII numbers such as '37' or '42'. This is bad practice, since most readers do not know ASCII encoding off by heart so it is unclear to the reader what represent. [sic]

```
- isDigit #"7";
> true : bool
```

   - `fun isDigit c = (Char.>= (c, #"0")) andalso (Char.<= (c, #"9"));`

     This function performs two comparisons on the character code of character `c` to evaluate if it lies in the range bounded by characters "`0`" and "`9`" inclusive, and returns the boolean result.
     While the operators >= and <= normally have infix status, when used in the format `Char.>=` they take the two arguments as a tuple which can be read left to right, with >= replacing the comma. This format is used to enforce the correct overloading of the operators without requiring "`open Char`" to be executed first.

4. If character c represents a digit then return its integer equivalent; otherwise return `~1`. Again, your answer should not hard-code ASCII numbers.

```
- digitValue #"a";
> ~1 : int
- digitValue #"7";
> 7 : int
```

   - `fun digitValue c = if isDigit c then ((ord c) - (ord #"0")) else ~1;`

     Using the function previously defined in Q3, this function first checks if the character is a digit before returning a value. The "integer equivalent" value is derived mathematically from the character code. As digits are represented as the range "`0...9`" in the Poly/ML character set, the value can be derived by subtracting the character code of "`0`" from the given character's code.

5. Convert a real `r` into a tuple of its value and integer equivalent.

```
- conv 99.99;
> (99.99,99) : real * int
```

   - `fun conv r = (r, Real.floor r);`

     While `floor` is not overloaded, it is called in this manner so that "`open Real`" does not need to be executed first. Also note the question specifies flooring the number, rather than 'true' rounding.

6. Implement the NAND function from the truth-table below. *Your implementation should not use the ML primitives* `not`, `andalso`, *or* `orelse`. *We are looking for you to write a program that is almost identical to the specification of the problem.*

X	Y	X NAND Y
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

- `fun NAND true true = false | NAND _ _ = true;`

  Using pattern matching alongside the "_" wildcard this function returns false only when "`true true`" is passed as an argument - which matches the specification. In all other cases it will output `true`.