Symbolic & Declarative Computing / Artificial Intelligence (COS5012-B)

October 25, 2016 Week 5 Lab1

LAB MANUAL: MORE ON FUNCTIONS

In this tutorial you will learn about

- Control Structures
- User defined operators
- Sections
- Use of where and let keywords
- Indentation

If then else

The syntax for if expressions is

```
if <condition> then <true-value> else <false-value>
```

`<condition>` is an expression which evaluates to a `Bool`. If the `<condition>` is True then the `<true-value>` is returned, otherwise the `<false-value>` is returned. Note that in Haskell `if` is an expression (which is converted to a value) and not a statement (which is executed) as in many imperative languages. As a consequence, the `else` is mandatory in Haskell. Since `if` is an expression, it must evaluate to a result whether the condition is true or false, and the else ensures this.

```
describeLetter :: Char -> String
describeLetter c =
    if c >= 'a' && c <= 'z'
        then "Lower case"
        else if c >= 'A' && c <= 'Z'
            then "Upper case"
            else "Not an Alphabet"
```

Guards

Guards are clean ifs ad can be used interchangeably with `if` expressions

```
describeLetter' :: Char -> String
describeLetter' c
   | c >= 'a' && c <= 'z' = "Lower case"
   | c >= 'A' && c <= 'Z' = "Upper case"
   | otherwise            = "Not an Alphabet"
```

- Order is important
- `otherwise` is just a fancy word for `True`

- Guards are powerful and anything done with pattern matching can be done with guards.

```
head'' :: [a] -> a
head'' xs
      | null xs    = error "list is empty"
      | otherwise = xs !! 0
```

Exercise

- Write the **factorial** function using `Guards`
- Write a **`max'`** function that returns the larger of two comparable entities.
- Write your own version of **`compare'`** function which returns an `Ordering`.

Writing functions in infix notation

Functions is Haskell are usually written in prefix notation which means the function name is followed by the arguments

```
func_name arg1 arg2
```

To convert a function into infix notation we need to put the function name in single black quote

```
arg1 ` func_name` arg2
```

Define your own operators

```
(//) a b = a `div` b
Hugs> 35 // 4 -- 8
```

Note: `div` is used for integer division

Let's write our own version of **`and`** operator

```
(&) False _ = False
(&) _ False = False
(&) _ _ = True
```

Exercise

- Define other Boolean operators (or, xor)

Defining operations on Characters

```
isLower :: Char -> Bool
isLower x = 'a' <= x && x <= 'z'

isUpper :: Char -> Bool
isUpper x = 'A' <= x && x <= 'Z'

isDigit :: Char -> Bool
isDigit x = ’0’ <= x && x <= ’9’

isAlpha :: Char -> Bool
isAlpha x = isLower x || isUpper x

digitToInt :: Char -> Int
digitToInt c | isDigit c = ord c - ord ’0’

intToDigit :: Int -> Char
intToDigit d | 0 <= d && d <= 9 = chr (ord ’0’ + d)

toLower :: Char -> Char
toLower c | isUpper c = chr (ord c - ord ’A’ + ord ’a’)
| otherwise = c

toUpper :: Char -> Char
toUpper c | isLower c = chr (ord c - ord ’a’ + ord ’A’)
| otherwise = c
```

Sections

Sections are a nifty piece of syntactical sugar that can be used with operators. An operator within parentheses and flanked by one of its arguments

```
Hugs> (2+) 3
5
Hugs> (*4) 5
20
```

Use of `where` and `let`

- Haskell variables are immutable.
- Once defined, they can't change.

```
bmiTell :: (RealFloat a) => a -> a -> String
bmiTell weight height
    | weight / height ^ 2 <= 18.5 = "Skinny!"
    | weight / height ^ 2 <= 25.0 = "Normal!"
    | weight / height ^ 2 <= 30.0 = "Fat!"
    | otherwise = "Be careful about your health!"
```

Notice that we repeat ourselves here three times. Repeating yourself (three times) while programming is about as desirable as getting kicked in head. Since we repeat the same expression three times, it would be ideal if we could calculate it once, bind it to a name and then use that name instead of the expression. Well, we can modify our function like this:

```
bmiTell :: (RealFloat a) => a -> a -> String
bmiTell weight height
    | bmi <= 18.5 = "Skinny!"
    | bmi <= 25.0 = "Normal!"
    | bmi <= 30.0 = "Fat!"
    | otherwise   = "Be careful about your health!"
    where bmi = weight / height ^ 2
```

We put the keyword `where` after the guards (usually it's best to indent it as much as the pipes are indented) and then we define several names or functions.

```
bmiTell :: (RealFloat a) => a -> a -> String
bmiTell weight height
    | bmi <= skinny = "Skinny!"
    | bmi <= normal = "Normal!"
    | bmi <= fat    = "Fat!"
    | otherwise     = " Be careful about your health!!"
    where bmi = weight / height ^ 2
          skinny = 18.5
          normal = 25.0
          fat = 30.0
```

The names we define in the where section of a function are only visible to that function.

```
slope (x1,y1) (x2,y2) = let dy = y2-y1
                            dx = x2-x1
                        in dy/dx
```

Exercise

1. Write the above mentioned **`slope`** function with `where` instead of `let`
2. Write a function **`reciproc`** to calculate reciprocal of a given number.

   `reciprocal of x = 1/x`
3. Write a function **`abst`** to return absolute of an integer by using conditional expressions and guards.
4. Write a function **`signNumb`** to determine the sign an input number. The rules will be
   a. If the number is less than zero, it will return -1
   b. If the number is greater than zero, it will return 1
   c. If the number is equal to zero, it will return zero

Provide two definitions of this function, one using guards and other conditional expressions.

5. Write a function **`threeDifferent`** which will take three integers as input and determines whether they are equal are not. It will return True or False depending upon the result.
6. Write down a function **`maxOfThree`** which will take three integers as input values and find out the maximum number.
7. Write down a function **`intToStr`** which will take an integer as input in the range 1 to 5 and write down the corresponding string value. For example, for 1, it will return “One”. In case, the value is not in the specified range, an appropriate message should be displayed.

Whitespaces

- In Haskell, indentation matters.
- Code which is part of some expression should be indented further in than the beginning of that expression
- Use whitespaces instead of tabs

References

- https://en.wikibooks.org/wiki/Haskell/
- http://learnyouahaskell.com/syntax-in-functions