CISC 360 Assignment 1
due Wednesday, 2021–09–22 at 11:59pm, via onQ

Jana Dunfield

September 14, 2021

Reminder: All work submitted must be your own. You may (and should) ask for help from the instructor and/or TAs.

Most of the questions on this assignment are about writing code, but a few questions are about stepping. Please write your answers within the comments indicated in the file `a1.hs` (search for `Q3.1` and `Q3.2`).

Start early: Even if some questions look easy to you, Haskell is a very different language and you may need more time than you think.

Late policy: Assignments submitted up to 24 hours late (that is, by 11:59 pm the following day) will be accepted **without penalty**. Assignments submitted more than 24 hours late will **not** be accepted except with an accommodation or a consideration granted according to policy.

IMPORTANT: Your file must compile

Your file **must** load (`:load` in GHCi) successfully, or we will deduct **30%** from your mark.

If you are halfway through a problem and run out of time, **comment out the code that is causing :load to fail** by surrounding it with {- ... -}, and write a comment describing what you were trying to do. We can often give partial marks for evidence of progress towards a solution, but **we need the file to load and compile**.

Document your code

This is *mostly* a "fill-in-the-blanks" assignment, so you will not need to document much, unless you declare new functions. In that case, you need to write a comment that explains what your function does.

Strive for simplicity

You should try to find a simple solution. You do not have to find the simplest solution to get full marks, but you should not have an excessively complicated solution. Marks may be deducted if your solution is too complicated. If you are worried about whether your solution is too complicated, contact the instructor.

Be careful with library functions

Haskell has a rather large built-in library. This assignment is not about how to find library functions, but about how to use some of the core features of Haskell. You will not receive many marks if you just call a library function that solves the whole problem. The point is to solve the problem yourself.

If you are not sure whether you are calling a library function that solves the whole problem, contact the instructor. Note that if we *suggest* a library function, you may certainly use it.

(The only way I know to avoid this issue is to craft problems that are complicated and arbitrary, such that no library function can possibly solve them. I don't like solving complicated and arbitrary problems, and you probably don't either.)

1 Add your student ID

The file `a1.hs` will not compile until you add your student ID number by writing it after the `=`:

```
-- Your student ID:
student_id :: Integer
student_id =
```

You don't need to write your name. When we download your submission, onQ includes your name in the filename.

2 Writing and testing small Haskell functions

2.1 `between`

This function takes three integers `m`, `n` and `p`, and should return `True` if $m < n < p$, and return `False` otherwise.

Fill in the definition of `between` by replacing the word `undefined` with appropriate Haskell code, and, if necessary, deleting the `=` (for example, if you decide to define `between` using guards).

We have already written five test cases for `between`, called `test_between1`, `2`, etc. These are combined into `test_between`, which is `True` only if all of these tests pass.

2.2 `parity`

The `parity` function takes two integers. It should return 1 if exactly one of them is odd, and 0 otherwise (if both are odd, or both are even).

3 Stepping questions

3.1 First expression

These questions ask you to step two small expressions. For example, the first stepping question asks you to do the three steps needed to get the result. Replace the `_____` parts. (If you want to check the last line, you can find it by entering the expression into GHCi.)

```
   (\z -> z - (3 + z)) 10
=> ______________________    ________________________
=> ______________________    ________________________
=> __                        by arithmetic
```

3.2 Second expression

The other expression is somewhat larger, and uses a function `ten`, which is defined in the comment.

Hint: The first step does *not* apply (call) the function `ten`. Haskell sees the expression

```
(\q -> (\y -> q 5)) ten 2
```

as the expression

```
(\q -> (\y -> q 5)) ten
```

applied to the argument 2.

Hint: A correct solution steps 4 times (as indicated by the structure of blanks).

4 spiral

In this question, you need to write a function `spiral`. Your function should be *recursive*—its definition will call itself. Follow the specification given:

```
Q4.
  Write a function ‘spiral’ that, given a pair of numbers ‘span’ and ‘dir’,
  returns 1 if ‘span’ equals 0,
  and otherwise returns (span * dir) * spiral (span - 1, 0 - dir).
```

Hint: Most of the code you need is already in the above specification.

5 spiral_seq

In this question, you'll need to return Haskell strings, which are lists of characters. We have not discussed lists in any detail; however, for this problem, you should only need to know the following:

- To return a string containing a single character, say `0`, write

  ```
  ['0']
  ```

  You could also write `"0"`.

- To concatenate (append) two strings, use the built-in function `++`. For example:

  ```
  ['1', '2', 3'] ++ ['4']
  ```

  returns `"1234"`, which is the same thing as `['1', '2', '3', '4']`.

- To convert an integer to its string (in decimal), call the built-in function `show`:

  ```
  (show 360)
  ```

  returns the string `"360"`.