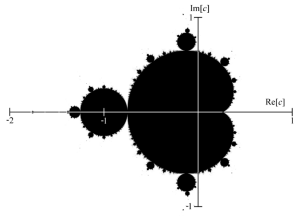
Computer Science 320 (Fall, 2009)
Concepts of Programming Languages

Assignment 1: Recursive Functions and Lists

Out: Thursday, September 3, 2009
Due: Monday, September 14, 2009

In this assignment, you will write two programs that plot ASCII text approximations of the Mandelbrot set: one in Haskell, and one in C.

Figure 1: Illustration of the Mandelbrot set.



The portion of your solution in Haskell should be defined within a module named `Mandelbrot`, and you should submit the file `Mandelbrot.hs` (or `Mandelbrot.lhs`). **File names are case sensitive.** The portion of your solution in C should be defined within a file named `mandelbrot.c`. Verbal responses to non-programming questions should be in the form of comments in your code.

Note: The solution to each part of the Haskell problems is meant to be extremely short (one to four lines). You may (and should) use functions from earlier problems to solve parts of later problems. Unless otherwise specified, your solutions may utilize functions from the standard prelude as well as material from the lecture notes and textbook.

Problem 1. (25 pts)

(a) Define a recursive function `prefix` that accepts a positive integer `n` and a list `xs` as input, and returns a list containing only the first `n` elements in the input list. **This function must have a recursive definition, and may not use any library functions.**

(b) Define a recursive function `suffix` that accepts a positive integer `n` and a list `xs` as input, and returns the list of elements that remain after the first `n` elements are dropped from the front of the list. **This function must have a recursive definition, and may not use any library functions.**

(c) Define a function `split` that takes a positive integer `n`, an element `y`, and a list `xs`. The function should insert the specified element `y` after every `n` elements in the list. **This function should work even when applied to infinite lists.** The following example illustrates how this function should work:

```
split 2 'X' ['a', 'b', 'c', 'd'] = ['a', 'b', 'X', 'c', 'd', 'X', 'e']
```

Problem 2. (45 pts)

(a) Define a function `plane` that takes a single argument `r` and returns the list of *all* points on the cartesian plane of the form `(x/r, y/r)` where `x/r` is between -2 and 1 while `y/r` is between -1 and 1. The list of points should be ordered from left to right, from bottom to top. For example, `plane 1` should return:

```
[(-2.0, -1.0), (-1.0, -1.0), ( 0.0, -1.0), ( 1.0, -1.0),
 (-2.0,  0.0), (-1.0,  0.0), ( 0.0,  0.0), ( 1.0,  0.0),
 (-2.0,  1.0), (-1.0,  1.0), ( 0.0,  1.0), ( 1.0,  1.0)]
```

Note that the length of the list should grow as `r` grows. You are allowed to use list comprehensions.

(b) Consider the function $P_{(x,y)}$ defined as follows:

$$P_{(x,y)}(u, v) = (u^2 - v^2 + x,\ 2uv + y)$$

We define the *orbit* $O(x, y)$ of a point (x, y) to be an infinite list of items:

$$O(x, y) = \{(0, 0),\ P_{(x,y)}(0, 0),\ P_{(x,y)}(P_{(x,y)}(0, 0)),\ P_{(x,y)}(P_{(x,y)}(P_{(x,y)}(0, 0))),\ \ldots\}$$

In other words, the nth entry of the list $O(x, y)$ is the $P_{(x,y)}$ function composed with itself n times and then applied to $(0, 0)$. Define a Haskell function `orbit` that takes a single point `(x,y)` as an argument and returns an infinite list corresponding to $O(x, y)$. You may want to define a helper function corresponding to $P_{(x,y)}$.

(c) Define a recursive function `disp` that takes two arguments: a number `d` and a list of pairs. Every pair in this input list consists of a number followed by a character, and you can assume the input list is always in ascending order. For example, a possible input list might be:

```
[(0.15, '#'), (0.5, 'x'), (1, '.')]
```

The function `disp` should return the character from the list that corresponds to the smallest number on the list that is greater than the input `d`, and if `d` is larger than all the number in the list, `disp` should return a space character, `' '`. For example,

```
disp 0.01 [(0.15, '#'), (0.5, 'x'), (1, '.')]  =  '#'
disp 0.4  [(0.15, '#'), (0.5, 'x'), (1, '.')]  =  'x'
disp 100  [(0.15, '#'), (0.5, 'x'), (1, '.')]  =  ' '
```

The Mandelbrot set is defined to be the set of points `(x,y)` for which the orbit sequence of points does not escape to infinity but stays within a fixed distance from the origin.

One way to approximate the Mandelbrot set is to consider a certain element within the orbit of every point on the plane (such as the 12th element) and to check whether that element is within a certain fixed distance from the origin. You should use the following function to calculate distances of points from the origin:[1]

```
norm (x,y) = x*x + y*y
```

These distance values can then be used with `disp` and `orbit` to turn points on the plane into appropriate ASCII characters within an ASCII plot of the Mandelbrot set.

(d) Define a function `mandelbrot` that takes three arguments: `r` represents the resolution of the approximation (to be used with the `plane` function), `i` represents the index of the elements to check in the orbit lists of the points, and `l` represents the formatting list (to be used with the `disp` function). This function should return a list of characters (which is equivalent in Haskell to a string) that corresponds to a picture approximating the shape of the Mandelbrot set on the plane. You will need to combine the `split`, `plane`, `disp`, and `orbit` functions appropriately; list comprehensions are allowed.

Once you've defined the function `mandelbrot`, you can generate an ASCII version of an approximation of the Mandelbrot set by evaluating the expression:

```
putStr (mandelbrot 20 20 [(0.15, '#'), (0.5, 'x'), (1, '.')])
```

If your solution is correct, the output should unmistakably resemble Figure 1.

Problem 3. (30 pts)

(a) Implement a program in C that takes two positive integers as command line arguments and displays an ASCII text approximation of the Mandelbrot set using the resolution and orbit index specified by the integers. For each positive integer inputs `r` and `i`, the output of your C program should match the results of

```
putStr (mandelbrot r i [(0.15, '#'), (0.5, 'x'), (100, '.')])
```

(b) Briefly describe three points of contrast or comparison between the two solutions (in Haskell and in C). These could be about the kinds of issues you had to think about while writing the solutions or how language features and characteristics affected your solutions (e.g. in terms of quality, flexibility, or efficiency).

[1] Note for those who are curious: this function is preferable over the Euclidean metric because the built-in `sqrt` function does not behave well on extremely small double precision values. Ideally, this assignment would use the `Rational` type to represent numbers, but this is a more advanced topic.