CSE 341 Section Handout #4
Cheat Sheet

Curried Functions

```
fun name param1 param2 ... paramN = expression
```

Example:

```
(* post: computes x^y, y >= 0 *)
fun pow x 0 = 1
|   pow x y = x * pow x (y-1);
```

Currying Operators

Syntax:

```
val name = curry operator;
```

Example:

```
(* doubles a value *)
val double = curry op* 2;
```

Types

Syntax:

```
datatype name = value1 | value2 | ... | valueN;
datatype name = value1 of typeExpression | value2 of typeExpression | ... ;
```

Example:

```
datatype Fruit = Apple | Orange | Banana | Tomato;
datatype Beverage = Water
    |   Coffee of string * bool
    |   Wine of string * int
    |   Beer of string;
```

Binary Trees

```
datatype IntTree = Empty | Node of int * IntTree * IntTree;

fun add(Empty, n) = Node(n, Empty, Empty)
|   add(Node(data, left, right), n) =
   if n <= data then Node(data, add(left, n), right)
   else Node(data, left, add(right, n));

fun addAll([]) = Empty
|   addAll(first::rest) = add(addAll(rest), first);

fun height(Empty) = 0
|   height(Node(_, left, right)) = 1 +
        Int.max(height(left), height(right));
```

Option Types

```
NONE
SOME expression
```

CSE 341 Section Handout #4
Questions

1. Define a new data type named `Vehicle` that can be either a bicycle, car, horse, or pogo stick. A bicycle is represented by its number of gears (speeds). A car is represented by its make and model (strings) and year of manufacture (an integer). A horse is represented by its age (an integer) and gender, which is either male or female. A pogo stick does not require any parameters for its type constructor.

2. Define a function `horsepower` that accepts a vehicle and produces its "horse power" (hp).

a. Write an initial version of the function that produces a value for any kind of vehicle. A horse has exactly 1 horsepower, of course. A bicycle generates about 0.35 hp. A car whose make is Ford or Chevy has 0 horsepower (since it is always broken), but any other car has 500 horsepower. A pogo stick generates 10,000 horsepower.

b. Add a new type of vehicle called the transporter that teleports the user from one location to another. (The type constructor for a transporter has no parameters.) This vehicle does not have horsepower; modify your `horsepower` function to not produce any value when passed a transporter. Achieve this using an `int option` as your function's result type.

3. Define a function `contains` that takes a binary search tree of ints and a value *n* and produces whether or not the tree contains *n* (`true` if it does, `false` if it does not). The function should take advantage of the fact that the tree is a binary search tree (i.e., it should take time proportional to the height of the tree).

4. Define a function `max` that takes a binary search tree of ints and produces the maximum value stored in the tree. The return type should be `int option` because not all trees have a maximum value. You should take advantage of the fact that the tree is a binary search tree to quickly find the maximum.

5. Define a function `printTree` that takes a binary search tree of ints as a parameter and prints its elements to the console, one per line, using an in-order traversal (left, then root, then right). The empty tree should produce no output when it is printed. For extra challenge, try writing a version that also indents each node value sideways 4 spaces for each level deep it is in the tree.

6. Write a function `removeLeaves` that removes the leaves from a binary search tree of ints. A leaf is a node that has empty left and right subtrees. If your method is called on an empty tree, the method does not change the tree because there are no nodes of any kind (leaf or not).

7. Define a function `collapse` that takes a binary search tree of ints as a parameter and produces a sorted list containing the same ints.

8. Define a function `treeMap` that takes a function and a binary tree of ints as parameters and produces the binary tree that results from applying the function to every value in the tree. The resulting function will not be polymorphic because it will require a function of the form `int -> int`.

CSE 341 Section Handout #4
Solutions

1.

```
datatype Gender = Male | Female;
datatype Vehicle = Bicycle of int
                 | Car of string * string * int
                 | Horse of int * Gender
                 | PogoStick;
```

2.

```
(* a. *)
fun horsepower(Horse(_, _)) = 1.0
|   horsepower(Bicycle(_) = 0.35
|   horsepower(Car(make, _, _)) = case make of
                                     "Ford"  => 0.0
                                   | "Chevy" => 0.0
                                   | other   => 500.0
|   horsepower(Car(_, _, _)) = 500.0
|   horsepower(PogoStick) = 10000.0;

(* b. *)
datatype Vehicle = Bicycle of int
                 | Car of string * string * int
                 | Horse of int * Gender
                 | PogoStick
                 | Transporter;
fun horsepower(Horse(_, _)) = SOME 1.0
|   horsepower(Bicycle(_) = SOME 0.35
|   horsepower(Car(make, _, _)) = case make of
                                     "Ford"  => SOME 0
                                   | "Chevy" => SOME 0
                                   | other   => SOME 500
|   horsepower(Car(_, _, _)) = SOME 500.0
|   horsepower(PogoStick) = SOME 10000.0
|   horsepower(Transporter) = NONE;
```

3. Two possible solutions appears below.

```
fun contains(Empty, n) = false
|   contains(Node(data, left, right), n) =
        if n = data then true
        else if n < data then contains(left, n)
        else contains(right, n);

fun contains(Empty, _) = false
|   contains(Node(data, left, right), n) =
        n = data orelse (n < data andalso contains(left, n))
                 orelse contains(right, n);
```

4.

```
fun max(Empty) = NONE
|   max(Node(data, _, Empty)) = SOME data
|   max(Node(_, _, right)) = max(right);
```

CSE 341 Section Handout #4
Solutions

5.

```
fun printTree(Empty) = ()
|   printTree(Node(data, left, right)) = (
        printTree(left);
        print(Int.toString(data) ^ "\n");
        printTree(right);
    );

(* trickier version that indents the node values *)
fun printTree2(t) =
    let
        fun printHelper(Empty, _) = ()
        |   printHelper(Node(data, left, right), indent) = (
                printHelper(left,  indent ^ "    ");
                print(indent ^ Int.toString(data) ^ "\n");
                printHelper(right, indent ^ "    ")
            );
    in
        printHelper(t, "")
    end;
```

6.

```
fun removeLeaves(Empty) = Empty
|   removeLeaves(Node(data, Empty, Empty)) = Empty
|   removeLeaves(Node(data, left, right)) =
        Node(data, removeLeaves(left), removeLeaves(right));
```

7.

```
fun collapse(Empty) = []
|   collapse(Node(data, left, right)) =
    collapse(left) @ data::collapse(right);
```

8.

```
fun treeMap(f, Empty)= Empty
|   treeMap(f, Node(data, left, right)) =
   Node(f(data), treeMap(f, left), treeMap(f, right));
```