In [1]:
```
import $file.hw5stdlib
import hw5stdlib._
```

```
Compiling hw5stdlib.sc
```

Out[1]:
```
import $file.$

import hw5stdlib._
```

Homework 5

In this assignment we will develop some helpful structures for building the next interpreter.

Dictionaries

We will need to define the datatype `Dictionary` before we write the first interpreter for lettuce. We'll begin by giving the definition below:

$$\begin{aligned}\textbf{Dictionary}\ A\ B\ ::= &\ \text{EmptyDict} \\ \mid &\ \text{Entry}\ A\ B\ (\textbf{Dictionary}\ A\ B)\end{aligned}$$

This will create dictionaries of key-value pairs which will be very useful for implementing contexts in the next homework assignment. The definition of this type in Scala is given below:

In [2]:
```
sealed trait Dictionary[+A, +B]
case object EmptyDict extends Dictionary[Nothing, Nothing]
case class Entry[A,B](key : A, value : B, xs : Dictionary[A, B]) extends
```

Out[2]:
```
defined trait Dictionary
defined object EmptyDict
defined class Entry
```

For both of the following functions you will need to provide a parameter for an equality-checking function. This is so we can check if keys are equal to eachother

Membership(In) (5pts)

Write a function called `isIn` which returns `True` if the Dictionary contains a given key and `False` if it does not.

In [11]:
```
// BEGIN SOLUTION
def isIn[A,B](eq : ((A,A) => Bool), dict : Dictionary[A,B], k : A ) : Bo
    case EmptyDict        => False
    case Entry(k1, v, kvs) => eq(k,k1) match {
        case True  => True
        case False => isIn(eq, kvs, k)
    }
}
// END SOLUTION
```

Out[11]:
```
defined function isIn
```

In [13]:
```
assert(isIn(nat_eq, Entry(five, True, EmptyDict), five) == True)
assert(isIn(nat_eq, Entry(eight, True, Entry(five, True, EmptyDict)), fi
assert(isIn(nat_eq, Entry(five, True, EmptyDict), six) == False)
assert(isIn(nat_eq, EmptyDict, seven) == False)
passed(5)
```

```
*** Tests Passed (5 points) ***
```

Lookup (5 pts)

Write a function that retrieves a value that corresponds to a key in a dictionary. If the key is not in the map then return Nothing.

$$\texttt{lookup} : (a \to a \to \mathbb{B}) \to \texttt{Dict}\ a\ b \to a \to \texttt{Maybe}\ b$$

In [16]:
```
// BEGIN SOLUTION
def lookup[A,B](eq : (A,A) => Bool, dict : Dictionary[A,B], k : A) : Mayb
    case EmptyDict        => None
    case Entry(k1, v, kvs) => eq(k,k1) match {
        case True  => Just(v)
        case False => lookup(eq, kvs, k)
    }
}
// END SOLUTION
```

Out[16]:
```
defined function lookup
```

In [17]:
```
assert(lookup(nat_eq, Entry(five, "ham", EmptyDict), five) == Just("ham"
assert(lookup(nat_eq, Entry(eight, seven, Entry(five, three, EmptyDict))
assert(lookup(nat_eq, Entry(five, True, EmptyDict), six) == None)
assert(lookup(nat_eq, EmptyDict, seven) == None)
passed(5)
```

```
*** Tests Passed (5 points) ***
```

Practice using these functions (5 pts)

Part A

Create the following dictionary as a scala value using this definition:

```
{
    five: True,
    six: False,
    seven: False
}
```

Place it in a variable named " `ans_3a` "

In [21]:
```
// Solution should look like `val ans3 = ???`
// BEGIN SOLUTION
val ans_3a = Entry(five, True, Entry(six, False, Entry(seven, False, Emp
// END SOLUTION
```

Out[21]:
```
ans_3a: Entry[Succ, Product with Serializable with Bool] = Entry(Succ(S
ucc(Succ(Succ(Succ(Zero))))),True,Entry(Succ(Succ(Succ(Succ(Succ(Succ(Z
ero)))))),False,Entry(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Zero)))))))),Fa
lse,EmptyDict)))
```

In [22]:
```
assert(lookup(nat_eq, ans_3a, five) == Just(True))
assert(lookup(nat_eq, ans_3a, six) == Just(False))
assert(lookup(nat_eq, ans_3a, seven) == Just(False))
passed(2)
```

```
*** Tests Passed (2 points) ***
```

Part B

For the following, use the super secret `secret_dict` variable defined below. Use the functions you just defined to complete the following:

1. Look up the value mapped by key `Positive(one)` and place it in variable `ans_3b1`
2. Look up the value mapped by key `Positive(three)` and place it in variable `ans_3b2`
3. Check whether the key `Negative(three)` is in the map and place True or False in variable `ans_3b3`
4. Check whether the key `Negative(one)` is in the map and place True or False in variable `ans_3b4`

Note: For 1 and 2 remember to extract it from the maybe type, set the variable to the string `"does not exist"` if you get `None` instead of `Just(x)` .

In [23]:
```
val secret_dict = Entry(Positive(three), "abc",
                        Entry(Negative(one), "xyz",
                              Entry(Positive(one), "MIB", EmptyDict)))
```

Out[23]:
```
secret_dict: Entry[Product with Serializable with Integer, String] = En
try(Positive(Succ(Succ(Succ(Zero)))),abc,Entry(Negative(Succ(Zero)),xy
z,Entry(Positive(Succ(Zero)),MIB,EmptyDict)))
```

In [24]:
```
// BEGIN SOLUTION
val ans_3b1 = lookup(int_eq, secret_dict, Positive(one)) match {
    case Just(x) => x
    case None => "does not exist"
}

val ans_3b2 = lookup(int_eq, secret_dict, Positive(three)) match {
    case Just(x) => x
    case None => "does not exist"
}

val ans_3b3 = isIn(int_eq, secret_dict, Negative(three))

val ans_3b4 = isIn(int_eq, secret_dict, Negative(one))
// END SOLUTION
```

Out[24]:
```
ans_3b1: String = "MIB"
ans_3b2: String = "abc"
ans_3b3: Bool = False
ans_3b4: Bool = True
```

In [26]:
```
// Hidden tests (3 pts)
// BEGIN HIDDEN TESTS
assert(ans_3b1 == "MIB")
assert(ans_3b2 == "abc")
assert(ans_3b3 == False)
assert(ans_3b4 == True)
passed(3)
// END HIDDEN TESTS
```

```
*** Tests Passed (3 points) ***
```