

CS496: Special Assignment 1

April 5, 2021
Due: 8 April, 11:59pm

1 Assignment Policies

Collaboration Policy. This assignment must be completed **individually**. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

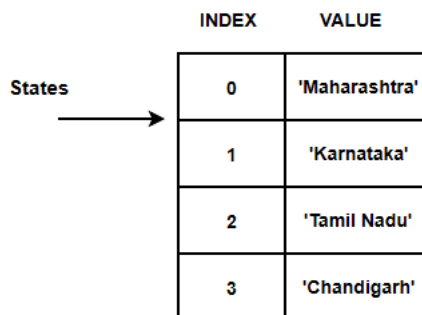
Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

2 Assignment

This assignment consists of extending REC to allow for the hash table data structure and its respective operations. The resulting language is dubbed RECHT

A hash table is a data structure where an associated key maps to a specified value. Often times, a hash function computes the index into an array of buckets from where the value can be found. For this assignment, the creation of a hash function to create the index of each key-value pairing is not necessary.



	INDEX	VALUE
States →	0	'Maharashtra'
	1	'Karnataka'
	2	'Tamil Nadu'
	3	'Chandigarh'

Figure 1: Depiction of a hash table data structure

The key to value pairing of a hash table makes it tremendously powerful since this facilitates an $O(1)$ lookup time for a specific key. For this assignment, the *insert*, *lookup*, *remove*, *size*, and, *empty?* operations will be implemented.

1. Emptytable: Creates an empty hash table.

```
1 utop # interp "emptytable";;
2 - : exp_val Recht.Ds.result = Ok (TableVal [])
```

2. Insert: Adds a key-value pair into the hash table.

```
1 utop # interp "insert(0, 1234, insert(1, 5678, insert(2,9101, insert
    ↪ (3, 1121, emptytable))))";;
2 - : exp_val Recht.Ds.result =
Ok
4 (TableVal
  [(NumVal 0, NumVal 1234); (NumVal 1, NumVal 5678); (NumVal 2,
    ↪ NumVal 9101);
6   (NumVal 3, NumVal 1121)])
```

Each key-value pairing is shown within its own set of parenthesis.

3. Lookup: Given a key, retrieve the value associated with the key in the hash table.

```
1 utop # interp "lookup(2, insert(0, 1234, insert(1, 5678, insert
    ↪ (2,9101, insert(3, 1121, emptytable))))";;
2 - : exp_val Recht.Ds.result = Ok (NumVal 9101)
```

If the key is not found, return an error indicating that the operation failed. To expedite the grading process, please return "Lookup: Key not found."

4. Remove: Given a key, remove the key-value pair from the hash table and display the new hash-table.

```
1 utop # interp "remove(2, insert(0, 1234, insert(1, 5678, insert
    ↪ (2,9101, insert(3, 1121, emptytable))))";;
2 - : exp_val Recht.Ds.result =
Ok
4 (TableVal
  [(NumVal 0, NumVal 1234); (NumVal 1, NumVal 5678); (NumVal 3,
    ↪ NumVal 1121)])
```

If the key does not exist, return an error. To expedite the grading process, please return "Remove: key not found."

5. Size: Returns the amount of elements in the hash table.

```
1 utop # interp "size(remove(2, insert(0, 1234, insert(1, 5678, insert
    ↪ (2,9101, insert(3, 1121, emptytable))))";;
- : exp_val Recht.Ds.result = Ok (NumVal 3)
```

6. Empty: Returns a boolean indicating whether the hash table is populated or not.

```
utop # interp "empty?(remove(2, insert(0, 1234, insert(1, 5678, insert
  ↪ (2,9101, insert(3, 1121, emptytable))))))";
2 - : exp_val Recht.Ds.result = Ok (BoolVal false)
```

3 Implementing hash tables in REC

To facilitate the process of implementing hash tables in REC, a stub has been provided for you in Canvas. This stub has been obtained by taking the interpreter for REC and applying some changes. Here is a summary of the changes:

1. The parser.mly file has been updated so that the parser is capable of parsing expressions such as:

```
utop # parse "lookup(1, (remove(2, insert(0, 1234, insert(1, 5678,
  ↪ insert(2,9101, insert(3, 1121, emptytable))))))";;
```

The result of parsing this expression would be:

```
- : expr =
2 Lookup (Int 1,
  Remove (Int 2,
4    Insert (Int 0, Int 1234,
      Insert (Int 1, Int 5678,
6        Insert (Int 2, Int 9101, Insert (Int 3, Int 1121, EmptyTable))))))
```

2. Note the new additions to ast.ml:

```
type expr =
2   | Var of string
  | Int of int
4   | Add of expr*expr
  | Sub of expr*expr
6   | Mul of expr*expr
  | Div of expr*expr
8   | Let of string*expr*expr
  | IsZero of expr
10  | ITE of expr*expr*expr
  | Proc of string*expr
12  | App of expr*expr
  | Letrec of string*string*expr*expr
14  | Pair of expr*expr
  | Fst of expr
16  | Snd of expr
  | EmptyTable
18  | Insert of expr*expr*expr
  | Lookup of expr*expr
20  | Remove of expr*expr
  | IsEmpty of expr
```

```
22 | Size of expr
    | Debug of expr
```

3. Note the new addition to ds.ml

```
1 let table_of_tableVal : exp_val -> ((exp_val*exp_val) list) ea_result
  ↪ = function
    | TableVal l -> return l
3   | _ -> error "Expected a hash table!"
```

4 Trying Out Your Code

We have provided a few test cases in the stub on Canvas. In the parent directory, run:

```
1 dune runtest
```

This will run the test cases that we have provided for you. Please note that these test cases are by no means exhaustive. We encourage you to thoroughly test your submission on edge cases. You can do so by typing out various commands in the interpreter as demonstrated in section 2.

5 Submission Instructions

Submit a file named SA1_<SURNAME>.zip through Canvas. Include all files from the stub.