Overview

In this assignment, we will Mob program an implementation of an abstract data type system in plain old C. In the process, we will a base abstract data type, and a string data type that “inherits” from that.

Individually, you will implement the hash table data type itself. The source code for this data type is the only thing you will need to hand in.

For extra credit, implement a Dictionary. Create a csAssocRef data type. It has a key and a value both of which are csAdtRef type. Redefine equals on the Assoc so that two assocs are equal if there key’s are equal. Implement methods on the dictionary at(self, key) and atPut(self, key, value).

Data Type Specifications

All abstract data types must implement the following three methods,

methods

// Return the hash code of self. By default a LFSR.
int hash(csAdtRef self)

// Return true if self = other
bool equals(csAdtRef self, csAdtRef other)

// Return a c string describing self.
char *description(csAdtRef self)

The String abstract data type:

constructor

// Return a string data type wrapped around the c string.
csStringRef newCsString(char *cstr)

methods

// Return the fnv-1a hash of the c string in self
int hash(csAdtRef self)

// make sure other is a csStringRef
// Return true if the c string inside self is equal to the c string inside other.
bool equals(csAdtRef self, csAdtRef other)

// Return my c string as my own description.
char *description(csAdtRef self)
The Hash Table abstract data type:

constructor

// Allocate a table big enough to hold capacity items.
```
csHashTableRef newCsHashTable(int capacity)
```

methods

// Return the number of items in the table.
```
int size(csHashTableRef self)
```

// Return true if item is in the table already.
```
bool includes(csHashTableRef self, csAdtRef item)
```

// If adding the item would make the table 3/4 full, grow the table.
// Add item to the table.
// If already present, overwrite old value.
// Return true if successful.
```
bool add(csHashTableRef self, csAdtRef item)
```

// This next one is trickier
// Remove item from the table if present.
// If item was present, return it, else return NULL.
```
csAdtRef remove(csHashTableRef self, csAdtRef item)
```

private methods (don't need to be in the dispatch table)

// return the index of item, or the index of the first empty slot
```
int scanFor(csHashRef self, csAdtRef item)
```

// allocate a table twice as big as the current one,
// add all of the items in the old table to the new one
```
bool grow(csHashRef self)
```