Lab2 - Documentation for the Symbol Table

The “SymbolTable” type is a structure containing an array of pointers to “Item” entities(explained below), and an unsigned integer denoting its size.

```
type SymbolTable struct {
    items []*Item
    size uint
}
```

The “Item” type is a structure representing an entity stored in the SymbolTable. It contains the key and value, which are strings, the hash code of each individual entity, and a pointer to the next item.

```
type Item struct {
    Key      string
    Value    string
    HashCode uint
    Next     *Item
}
```

The “hashFunc” package-private procedure computes the hash code of a string given as parameter by adding up the ASCII code of each individual character in the string.

```
func hashFunc(val string) (hashCode uint) {
    hashCode = 0

    for _, ch := range val {
        hashCode += uint(ch)
    }

    return
}
```

The public procedure “New” acts as a constructor of the “SymbolTable” type, abstracting away unnecessary details about its initialization.

```
func New() *SymbolTable {
    return &SymbolTable{
        items: make([]*Item, 10),
        size:  10,
    }
}
```

The “Add” public method of the SymbolTable struct creates a new Item object with the key and value provided as parameters, and inserts it into the SymbolTable. The collisions are solved via chaining.

```
func (st *SymbolTable) Add(k, v string) {
    i := st.getIndex(k)
```

```
    hc := hashFunc(k)

    head := st.items[i]

    for head != nil {
        if head.Key == k && head.HashCode == hc {
            head.Value = v
            return
        }
        head = head.Next
    }

    st.size++

    head = st.items[i]
    st.items[i] = &Item{
        Key:      k,
        Value:    v,
        HashCode: hc,
        Next:     head,
    }

    if float64(st.size) >= 0.7 {
        temp := st.items

        st.items = make([]*Item, 2*st.size)

        st.size = 2 * st.size

        for _, x := range temp {
            for x != nil {
                st.Add(x.Key, x.Value)
                x = x.Next
            }
        }
    }
}
```

The “Remove” public method deletes an element from the “SymbolTable” which contains key “k”, returning its stored value and a nil error on success, and an empty string and a non-nil error if the element is not found.

```
func (st *SymbolTable) Remove(k string) (string, error) {
    i := st.getIndex(k)
    hc := hashFunc(k)

    head := st.items[i]

    var prev *Item

    for head != nil {
```

```
            if head.Key == k && hc == head.HashCode {
                break
            }
            prev = head
            head = head.Next
        }

        if head == nil {
            return "", errors.New("not found")
        }

        st.size--

        if prev != nil {
            prev.Next = head.Next
        } else {
            st.items[i] = head.Next
        }
        return head.Value, nil
    }
```

The “Get” public method retrieves the value stored in the “Item” with key “k”; in the case that the element is not found, an empty string will be returned alongside a non-nil error.

```
    func (st *SymbolTable) Get(k string) (string, error) {
        i := st.getIndex(k)
        hc := hashFunc(k)

        head := st.items[i]
        for head != nil {
            if head.Key == k && head.HashCode == hc {
                return head.Value, nil
            }
            head = head.Next
        }
        return "", errors.New("not found")
    }
```

The “getIndex” package-private method computes the index of the element with key “k” in the “SymbolTable”, returning it as an unsigned integer.

```
    func (st *SymbolTable) getIndex(k string) (i uint) {
        hc := hashFunc(k)

        i = hc % st.size

        return
    }
```