CSC 120 **ICA-33**

Work with your neighbor. (This will be graded for participation only.)

1. The solution for the Dictionary ADT from last time is below:

```
class Dictionary:

    def __init__(self,capacity):

        # each element will be a key/value pair

        self._pairs = [None] * capacity

        self._nextempty = 0

    def put(self, k, v):

        self._pairs[self._nextempty] = [k,v]

        self._nextempty += 1

    def get(self, k):

        for pair in self._pairs[0:self._nextempty]:

            if pair[0] == k:

                return pair[1]

        return None
```

Modify the ADT above to use a hash function to compute the index for a new key/value pair. Use the following hash function:

```
def _hash(self, k):

       return len(k) % len(self._pairs)
```

Then in `put()` and `get()` use this:

```
index = self._hash(k)
```

Answer is on next page.

ANS:

```
class Dictionary:
    def __init__(self, capacity):
        # each element will be a key/value pair
        # represented as a list
        self._pairs = [None] * capacity

    def _hash(self, k):
        return len(k) % len(self._pairs)

    def put(self, k, v):
        self._pairs[self._hash(k)] = [k,v]

    def get(self, k):
        return self._pairs[self._hash(k)][1]

    def __str__(self):
        return str(self._pairs)
```

2. Use open addressing with linear probing to insert the key 23 into the hash table below. Give the probe sequence.

   The hash function is: `hash(key)= key % 7`

0	1	2	3	4	5	6
14	24	2	10		19	

ANS:

0	1	2	3	4	5	6
14	24	2	10		19	**23**

Probe sequence is**: 2, 1, 0, 6**

3. Modify the `put()` method of the Dictionary ADT below to use open addressing with linear probing.

```
class Dictionary:

    def __init__(self, capacity):

       # each element will be a key/value pair

        self._pairs = [None] * capacity

    def _hash(self, k):

        return len(k) % len(self._pairs)

    def put(self, k, v):

        self._pairs[self._hash(k)] = [k,v]
```

Write the new `put()` method below:

Answer is on next page.

ANS:

# This code assumes the hash table will never be full and that the key

# is being placed in the hash table for the first time.

```
def put(self, k, v):
    i = self._hash(k)
    if self._pairs[i] != None:
        while True:
            i -= 1
            if i < 0:
                # alternatively, use negative indexes
                i = len(self._pairs) - 1
            if self._pairs[i] == None:
                break

    self._pairs[i] = [k,v]
```