

## Computer Science I – Exercise Hash Tables (Solutions)

1) Consider a hash table that uses the linear probing technique with the following hash function  $f(x) = (5x+4)\%11$ . (The hash table is of size 11.) If we insert the values 3, 9, 2, 1, 14, 6 and 25 into the table, in that order, show where these values would end up in the table?

index	0	1	2	3	4	5	6	7	8	9	10
value	25	6		2		9			3	1	14

2) Do the same question as above, but this time use the quadratic probing strategy.

index	0	1	2	3	4	5	6	7	8	9	10
value		14	6	2		9	25		3	1	

3) Do the question above, but draw a picture of what the hash table would look like if separate chaining hashing was used.

Index

0:

1: 6

2:

3: 2

4:

5: 9

6:

7:

8: 25 -> 14 -> 3

9: 1

10:

4) Edit the code inhtablelinear.c so that quadratic probing is the searching strategy used.

Changes are denoted in bold and underlined below. //there can be other solution too!

```
// Pre-condition: h points to a valid hash table that IS //  
not yet half full.  
// Post-condition: word will be inserted into the table h.  
void insertTable(struct htable *h, char word[]) {
```

```
    int hashval;  
    hashval = hashvalue(word);  
  
    // Here's the quadratic probing part.  
    int i = 1;  
    while (strcmp(h->entries[hashval], "") != 0) {  
        hashval = (hashval+i)%TABLE_SIZE;  
        i *= 2;  
    }  
    strcpy(h->entries[hashval], word);  
}
```

```
// Pre-condition: h points to a valid hash table that is no  
// more than half full.  
// Post-condition: 1 will be returned iff word is stored in  
the table pointed to  
// by h. Otherwise, 0 is returned.  
int searchTable(struct htable *h, char word[]) {
```

```
    int hashval;  
    hashval = hashvalue(word);  
  
    // See what comes first, the word or a blank spot.  
    int i = 1;  
    while (strcmp(h->entries[hashval], "") != 0 &&  
        strcmp(h->entries[hashval], word) != 0) {  
        hashval = (hashval+i)%TABLE_SIZE;  
        i *= 2;  
    }
```

```
    // The word was in the table.  
    if (strcmp(h->entries[hashval], word) == 0)
```

```

        return 1;

// It wasn't.
return 0;

}

// Pre-condition: h points to a valid hash table that is no
// more than half full.
// Post-condition: deletes word from the table pointed to
// by h, if word is
// stored here. If not, no change is made
// to the table pointed
// to by h.
void deleteTable(struct htable *h, char word[]) {

    int hashval;
    hashval = hashvalue(word);

    // See what comes first, the word or a blank spot.
    int i = 1;
    while (strcmp(h->entries[hashval], "") != 0 &&
           strcmp(h->entries[hashval], word) != 0) {
        hashval = (hashval+i)%TABLE_SIZE;
        i *= 2;
    }

    // Reset the word to be the empty string.
    if (strcmp(h->entries[hashval], word) == 0)
        strcpy(h->entries[hashval], "");

    // If we get here, the word wasn't in the table, so
    nothing is done.
}

```