Computer Science I –Exercise
Hash Tables

$(5 \cdot 3) + 4 \% 11$
$= 15 + 4 \% 11$
$= 19 \% 11$
$= \boxed{8}$
$(5 \cdot 14) + 4 \% 11$
$= 8 + 1 = 9$
$9 + 1 = \boxed{10}$

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

$(5 \cdot 14) + 4 \% 11$
$= 8 + 1^2 = 9 \leftarrow$ taken
try $8 + 2^2 = 12 \% 11 = \boxed{1} \leftarrow$ free

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

$(5 \cdot 6) + 4 \% 11$
$= 1 \leftarrow$ taken
try $1 + 1^2 = \boxed{2} \leftarrow$ free

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

$(5 \cdot 25) + 4 \% 11$
$= 8 \leftarrow$ taken
$\left.\begin{array}{l} 8 + 1^2 \\ 8 + 2^2 \end{array}\right]$ taken
$8 + 3^2 \% 11 = \boxed{6} \leftarrow$ free

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

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

note: index 8:
values: 3 ← 14 ← 25

4) Edit the code in htablelinear.c so that quadratic probing is the searching strategy used. You will need to modify insert function, then search and then delete. Add the code to your pdf when submitting.

code included below question 5

5) No need to submit, but you can practice: Try to edit this code so that it uses a dynamically sized array instead of a statically sized one. If you have extra time, use this code to read in a whole dictionary from a file and count how many places have to be checked on average before a word is found or determined to not be in the dictionary.

```
void insertTable(htable* h, char word[])
{
  int hashval = hashvalue(word);
  int i = 1; //start with 1^2...

  while (strcmp(h->entries[hashval], "") != 0)
  {
    hashval = (hashval+i)%TABLE_SIZE; //hashval+i instead of hashval+1...
    i *= 2; //i^2 becomes (i+1)^2...
  }

  strcpy(h->entries[hashval], word);
}

int searchTable(htable* h, char word[])
{
  int hashval = hashvalue(word);
  int i = 1; //start with 1^2...

  while (strcmp(h->entries[hashval], "") != 0 &&
  strcmp(h->entries[hashval], word) != 0)
  {
    hashval = (hashval+i)%TABLE_SIZE; //hashval+i instead of hashval+1...
    i *= 2; //i^2 becomes (i+1)^2...
  }

  if (strcmp(h->entries[hashval], word) == 0)
    return 1;

  return 0;
}

void deleteTable(htable* h, char word[])
{
  int hashval = hashvalue(word);
  int i = 1; //start with 1^2...

  while (strcmp(h->entries[hashval], "") != 0 &&
  strcmp(h->entries[hashval], word) != 0)
  {
    hashval = (hashval+i)%TABLE_SIZE; //hashval+i instead of hashval+1...
    i*= 2; //i^2 becomes (i+1)^2...
  }
  if (strcmp(h->entries[hashval], word) == 0)
    strcpy(h->entries[hashval],"");
}
```