Tree vs Bubble Sort

Jacob Tucker

CMPS 390

4/27/23

For this assignment I create 3 hash tables to store names from a file *input.txt*.

I define constants for each hash table to reuse their values throughout the code. I create 4 functions to make the program more modular.

The first function is *genHashTableIndex* which takes in a *nameHash* and *tableSize* and creates a table of that size with a gap of 10 between each hash index to allow room for 10 collisions each. This was decided on because the maximum number of collisions for the given data and minimum table size never exceeded 10. Also, 10 is an easy number to work with to create the gaps. I then subtract 3,510 from the values because 351 is the lowest *nameHash* generated and this will ensure that the hash indices begin at index 0. Then I use modular arithmetic to ensure the hash indices never exceed the given table size.

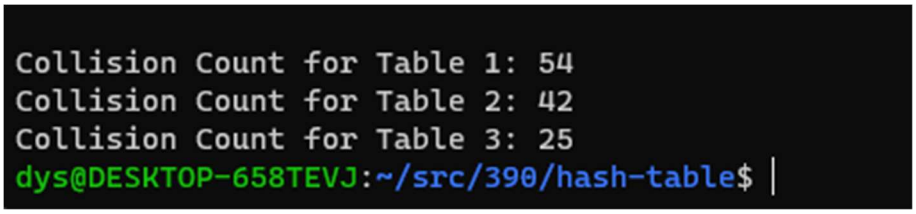
The *insertName* function takes in a pointer to the hash table, which itself is an array of strings, the index to insert the name at, and the name as arguments. It returns an integer representing whether a collision was detected or not. To detect this, since the hash tables are initialized to have NULL values at all indices, it simply looks to see if the given index is NULL or not. If not, then a name has already been inserted there so it must check the next index. Once it finds a NULL value at an index, it allocates memory for the name and stores it there.

The *showTable* function prints the index and name for all indices with a value other than NULL in the given table.

The *freeTable* function iterates through the given table freeing any memory that has been allocated by checking for NULL values.

```c
int main() {
  char* hashTable1[HASH1SIZE] = {NULL};
  char* hashTable2[HASH2SIZE] = {NULL};
  char* hashTable3[HASH3SIZE] = {NULL};

  int collision1Count = 0;
  int collision2Count = 0;
  int collision3Count = 0;

  char fileName[] = "input.txt";
  char line[30];
  FILE* file;
  file = fopen(fileName, "r");
  while (fgets(line, sizeof(line), file)) {
    if (line[strlen(line) - 1] == '\n') { line[strlen(line) - 1] = '\0'; }

    int nameHash = (line[0] - 'a') * (int)pow(26, 2) + (line[1] - 'a') * 26 + (line[2] - 'a');

    int index1 = genHashTableIndex(nameHash, HASH1SIZE);
    collision1Count += insertName(hashTable1, index1, line);

    int index2 = genHashTableIndex(nameHash, HASH2SIZE);
    collision2Count += insertName(hashTable2, index2, line);

    int index3 = genHashTableIndex(nameHash, HASH3SIZE);
    collision3Count += insertName(hashTable3, index3, line);
  }
  fclose(file);

  showTable(hashTable1, HASH1SIZE);
  showTable(hashTable2, HASH2SIZE);
  showTable(hashTable3, HASH3SIZE);
  printf("Collision Count for Table 1: %d\n", collision1Count);
  printf("Collision Count for Table 2: %d\n", collision2Count);
  printf("Collision Count for Table 3: %d\n", collision3Count);
  freeTable(hashTable1, HASH1SIZE);
  freeTable(hashTable2, HASH2SIZE);
  freeTable(hashTable3, HASH3SIZE);
}
```

In the *main* function I initialize the hash tables values to all NULL values and the collision counts for each to 0. I then read the input file line by line, calculating the hash for each name and then inserting it into each table as they are read. The outputs are then printed and the tables are freed. Here are the collision counts:



Here is the output for *showTable* being called with table 1:

```
dys@DESKTOP-658
0: dudz
1: anny
2: matt
10: joe
11: pam
12: gemini
20: barrack
21: larry
22: meriam
30: george
31: charles
32: issaac
33: chuck
34: ross
35: eric
36: ziggy
37: dewy
40: bill
41: billyjoe
42: johnson
43: max
50: dianna
51: raymond
52: huey
60: gertrude
61: webster
62: karl
63: karla
64: robert
65: junkun
70: howard
71: ghassan
72: rocky
73: clarence
80: dale
81: mitzee
90: tom
91: thomas
92: paul
93: kim
94: roy
95: sammy
100: bob
101: ellis
102: tena
103: kerry
104: ellie
105: dan
110: zack
111: greg
112: cris
113: francis
```

```
120: peter
121: halley
130: zeus
131: apollo
132: twirly
140: jerry
141: donna
150: brian
151: stewart
152: fred
160: bullwinkle
161: judy
162: lala
180: harry
181: mary
182: carl
183: marvin
184: hongkongfooey
190: theresa
191: ulyssess
192: avery
```

Here is the output for *showTable* being called with table 2:

```
0: dudz
1: anny
10: pam
20: meriam
30: chuck
31: ross
32: eric
33: ziggy
34: dewy
50: dianna
51: raymond
70: howard
71: ghassan
72: clarence
80: dale
90: thomas
91: paul
92: roy
100: bob
101: ellis
102: tena
103: kerry
104: ellie
105: dan
110: cris
130: apollo
140: jerry
141: donna
150: brian
160: bullwinkle
161: judy
162: lala
180: mary
181: carl
182: marvin
200: matt
210: joe
211: gemini
220: barrack
221: larry
230: george
231: charles
232: issaac
240: bill
241: billyjoe
242: johnson
243: max
250: huey
260: gertrude
261: webster
262: karl
263: karla
264: robert
```

```
265: junkun
270: rocky
280: mitzee
290: tom
291: kim
292: sammy
310: zack
311: greg
312: francis
320: peter
321: halley
330: zeus
331: twirly
350: stewart
351: fred
380: harry
381: hongkongfooey
390: theresa
391: ulyssess
392: avery
```

Here is the output for *showTable* being called with table 3:

```
0: anny
10: pam
30: eric
50: brian
51: huey
70: howard
80: mary
81: dale
82: marvin
83: mitzee
90: theresa
91: paul
100: tena
101: matt
102: dan
110: joe
130: zeus
131: issaac
132: chuck
140: johnson
141: max
160: bullwinkle
190: thomas
200: kerry
210: greg
240: donna
260: judy
261: robert
262: lala
270: rocky
271: clarence
300: dudz
310: zack
320: larry
330: ziggy
350: raymond
360: junkun
370: ghassan
380: carl
390: ulyssess
410: gemini
411: francis
420: meriam
421: peter
430: george
431: ross
440: jerry
450: fred
460: gertrude
490: kim
491: roy
510: cris
520: halley
530: apollo
```

```
531: twirly
532: dewy
540: bill
541: billyjoe
560: karl
561: karla
580: harry
590: tom
591: avery
600: bob
601: ellis
602: ellie
620: barrack
630: charles
650: stewart
651: dianna
660: webster
680: hongkongfooey
690: sammy
```

None of them seem very good for alphabetizing the names, though it does seem to get better as the table size increases. It's hard to see this for the 3 given sizes in the assignment, but if I increase the size of table 3 to 70,000 the ordering and grouping of names are noticeably improved.

```
0: anny
530: apollo
960: lala
1020: larry
1990: avery
3420: barrack
5440: bill
5441: billyjoe
6260: webster
6900: bob
7750: brian
7780: mary
7781: marvin
7800: matt
7840: max
8560: bullwinkle
8820: meriam
9880: mitzee
10180: carl
11830: charles
12030: chuck
12870: clarence
14510: cris
16880: dale
16900: dan
18030: dewy
18850: dianna
20540: donna
22000: dudz
25510: zack
26500: ellis
26501: ellie
26730: zeus
27630: ziggy
28010: pam
28030: eric
28090: paul
29120: peter
34710: francis
34750: fred
38210: gemini
38230: george
38260: gertrude
38870: ghassan
41510: greg
41650: raymond
```

Also, the smaller table size is more memory efficient since all 3 tables contain the same data. However, the collisions decreased as the table size increased. Overall, I believe the hash function should be improved to make the alphabetizing effective at more reasonable sizes.