OzoBlockly Editor—**Hash Table Program**—Teacher's Guide

Grades 11 – 12

by Richard Born, Associate Professor Emeritus, Northern Illinois University

This hash table lesson has been designed for students in grades 11 and 12 who have a passion for computer science. Using an example of storing student IDs and their corresponding names in a hash table, basic terms related to hash tables are investigated:

- Indexing
- Hashing techniques
- (Key, value) data pairs
- Collisions
- Linear probing
- Separate chaining
- Load factor

Next, your students study a complete Ozobot Editor blockly-based program that supports the basic hash table operations of adding a student to the table, searching the table for a student, and deleting a student. These three operations are all accomplished:

- using interactions with Ari, ***or***
- with Evo and interactions on the Ozobot Editor Terminal

It is especially "cool" to run the hash table program with Ari, as the student can hold Ari in his or her hand while entering information and receiving responses all on Ari.

Finally, the students are introduced to a simple program that makes use of an Ozobot LLM bot to learn more about hash tables and the capabilities of large language models.

If you haven't already done so, you are encouraged to look over the Student Guide. The share code for the hash table program, complete with detailed comments documenting the program is **um7yi7f.**

Programming Concepts Involved with Coding the Hash Table Program

- The use of variables to hold the values of program constants
- Subroutines to break up code into logical segments
- The use of loops to handle tasks that must be repeated several times
- The use of conditionals, as needed, to control the flow of logic
- Subroutine and variable names that support self-documentation
- Nesting of structures

Applicable CSTA (Computer Science Teachers Association) Standards

2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
2-AP-19	Document programs in order to make them easier to follow, test, and debug.
3A-DA-11	Create interactive data visualizations using software tools to help others better understand real-world phenomena.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3B-CS-02	Illustrate ways computing systems implement logic, input, and output through hardware components.
3B-AP-08	Describe how artificial intelligence drives many software and physical systems.
3B-AP-14	Construct solutions to problems using student-created components, such as procedures, modules and/or objects.

Answers to "Working with the Hash Table Program" Questions

Part 1

4. Greg's studentID hashes to index 5. Since index 5 is already in use by Diane, Greg would be stored at the next available index, which is 6.
5. Ed's studentID hashes to index 7. Since index 7 is empty, his data will be stored at inex 7.
6. Gloria's studentID hashes to index 5. Gloria's data will be stored at index 8 since it is the next available index after index 5.
7. Jim's studentID hashes to index 10. That is where it get stored, as index 10 is empty.
8. Carol's studentID hashes to index 8. However, 8 is already in use by Gloria, so Carol's data gets stored at the next available location, which is index 9.
9. Searching for both Mark and Ed results in found results.

10. A studentID of 467 will hash to index 9. Every item in the table is searched to see if the studentID there is 467. Since none of the studentID's are 467, the message "Not Found" is displayed.
11. Mark's studentID hashes to index 15, but Bev's data is there, so the next address is 16, but Denny's data is there. Sue's address is at index 0, so circularity of the table sets in an index 0 is scanned next to see if it is Mark's studentID. It is not. Then index 1 provides a match to Mark's ID. Mark is indicated as found. Then the user is given the option *Yes* to delete or *No* to not delete. The user selects *Yes* to delete Mark.
12. The entire table is searched for Mark's studentID, and is not found. So he was indeed deleted from the hash table.
13. There are currently 11 students in the table. The table size is 17. So the load factor is 11/17 ≈ 0.65 or 65%.

Part 2

2. You are informed that the "Table is Empty". You did not even need to key in Bev's studentID.
3. Again, you are informed that the "Table is Empty".
4. When you divide any studentID by 3, you will get a remainder of 0, 1, or 2. Therefore, possible indexes are 0, 1, or 2.
5. With the table currently full with Carol, Greg, and Gloria, there is no room for Rich. A message is displayed that the "Table is Full". There was no need to even enter Rich's studentID. Carol, Greg, and Gloria are stored at locations 2, 0, and 1, respectively.
6. Sue (408)is not found.
7. Greg (906) is found.
8. Gloria (923) is found.
9. Sue (408) is not found.

Part 3

3. There were no collisions.
4. There are 12 studens in the hash table. The size of the hash table is 47. Therefore, the load factor is 12/47 ≈ 0.26 or 26%. The number of collision seems to be inversely proprotional to th size of the hash table, provided that the load factor is not too high.