CST-201

17 November 2018

Green Group

Jackie Adair

Zachary Chambers

Dylan Dorn

In your group, propose one hash function for the numbers in the data on page 577 (the number is the key – index in the hash table array – and the name and number are the data). Define a hash table with 11 entries and describe a linear method for handling collisions between indexes when inserting data in the table. Demonstrate that the hashing function works, and collisions are handled, by storing and retrieving data from the hash table using the key as the position of a data item in the table.

WE created a HashedItem class to encapsulate the hash, search, storage, and print functions for this assignment. The class declaration is as follows:

The class defines a struct with two strings as elements: number, and name. (number is a string to allow ‘-‘’s to be passed to the hash function.

Constructor : initializes the array to 11 empty data structs.

FindByNumber: receives and returns a string. It searches the array for the passed number by hashing it and comparing the data->name with the same field in the hashed element of the array. If it is not a match, the function iterates over the elements until it finds it. If it reaches the end of the array, it wraps around to the 0th element and continues searching until it reaches the hashed element again. If still not found, it sends a false indicator and exits the function. If it finds it, it returns data->name.

PrintTable: iterates from the 0th element to the tableSize element and outputs the array’s ->name and ->number fields of each element.

Hash: accepts a string and returns an int. This function converts each char element of the passed string to its ASCII value using type casting, adds them all together, multiplies it by 7, then takes the absolute value of that product and modulus’s it by the tableSize (abs(sumASCII \* 7) % tableSize). This is the hash and is what is returned.

AddItem: This function takes two strings and returns a bool. The key string is hashed, then the array at the hash is checked to see if it is full. If it is not, the two strings are placed into that element of the array. If it is full, the linear probing begins and the next element of the array is checked until an empty element is found. The check wraps to the 0th element if the tableSize is reached and when an empty element is found, the strings are written. If the hash element is reached with no empty elements, a false is returned indicating a full table.

Driver: The main function creates a testItem and populates 11 elements of the array. A twelfth element is attempted but is not successful. The table is printed and a number is passed to the FindByNumber function which returns the name string for output.

