CS211 – Programming Practicum **Fall 2014**

Programming Project 8

Due: Friday, 10/17/13 at 11:59 pm

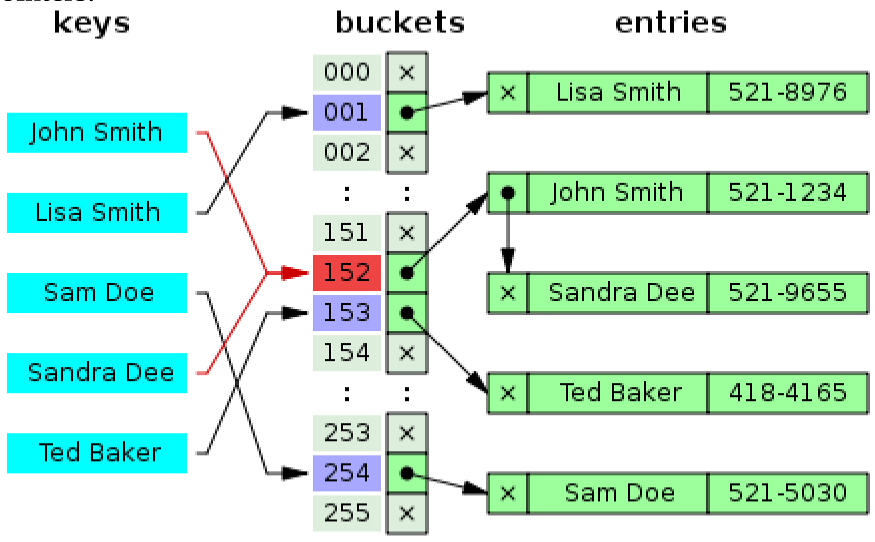
Hash Tables with Chaining

Hash tables are a great way to store lots of data in a program because they can perform operations like inserting a value, deleting a value, or looking up a value incredibly fast.

Hash Tables with Chaining use an array of linked lists, where each linked list only holds a portion of the overall data being stored. This will run fast if the values stored in the Hash Table are evenly distributed among all the linked lists and each list is pretty small. Each data value will be mapped with a specific position in the array and thus will be stored in the linked list at that position. So when a value is added to the hash table, first the position in the array must be determined. Once the position is known, the value is inserted into the linked list at that position. However, a Hash Table should not contain duplicates of a value, so once the position in the array is known, the value is only inserted if the list does not already contain that value.

The same idea is followed by deletes and look-ups. First determine the position of the array, then access the linked list at that position for the desired operation.

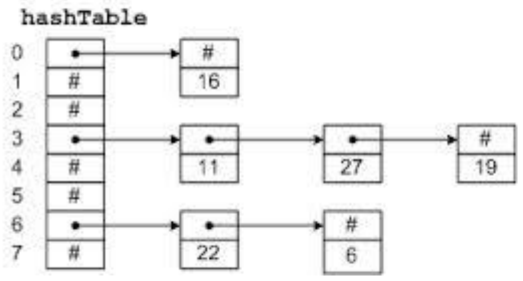
The following image shows a hash table with chaining that is storing names and phone numbers. The mapping uses the name values to determine the positions in that array where the data is stored (the exact mapping algorithm used for the example is not mentioned). Note that the names of "John Smith" and "Sandra Dee" both map to position 152 in the array. The "x" values are NULL pointers.



For this program, we will only be having integer values for our data. To determine which linked list will store an integer value, we will just find the remainder of the integer divided by the size of the array (i.e. value % size). The calculation that determines which position in the array is being access is called the "hash function". This project is using a very simple hash function. Often a much more complicated calculation is used for the hash function.

Below is another image showing a hash table with chaining that stores integer values. This example uses the same hash function that we will be using for our project, value % size. In this example, the size of the array is 8. The character # are NULL pointers.

- 16 is stored in the linked list at position 0 since 16 % 8 is 0
- 11 is stored in the linked list at position 3 since 11 % 8 is 3
- 27 is stored in the linked list at position 3 since 27 % 8 is 3
- 19 is stored in the linked list at position 3 since 19 % 8 is 3
- 22 is stored in the linked list at position 6 since 22 % 8 is 6
- 6 is stored in the linked list at position 6 since 6 % 8 is 6



Input for this Program

The input for the operations will come from standard input. The commands are to follow the descriptions given below. Note: that the form <int> could be any integer number and it will NOT be enclosed in angle brackets. <int> is just a notation to specify and integer value. The integer value is to be input on the same line as the command character. If the first character on the line is not one of the following characters, print an error message and ignore the rest of the information on that line.

q	- **quit** the program
i <int>	- **insert** the integer value into the hash table. The items in the list can be stored in any manner that you wish.
d <int>	- **delete** the integer value from the hash table. Be sure to properly deallocate the node after it is removed from the hash table.
c <int>	- display a message stating whether the given integer value is **contained** in the hash table.
r <int>	- **resize** the array to contain the number of positions indicated by the given integer value. You will need to re-hash all of the existing values in the hash table to new locations. Be sure to properly deallocate the memory no longer needed.
l	- **list** all the items contained in the hash table and include which array position they are located. First display all of the values (if any) in position 0 of the array, followed by all the values (if any) in position 1 of the array, followed by all the values (if any) in position 2 of the array, etc. This will allow us to verify that the hash table is being built correctly.

e	- **erase** all of the values currently stored in the hash table. Be sure to properly deallocate each node as it is removed from the hash table.
?	- list out the commands used for this program.

Initial Hash Table Size and Some Other Comments

Your program should start with a hash table of size 8. So the array would have positions 0 through 7 at the start. However, this will change whenever the user performs the resize "r" command.

Each linked list can store the values in any order you wish. You could always insert the values at the beginning or you could always insert the values at the end. The choice is up to the programmer. Some programmers like to insert the values in increasing order, since this may cut down on how many nodes in the linked list that must be inspected when deleting a value from the Hash Table or when determining if the Hash Tables contains a specific value.

The restaurant program from Lab 4 contains a similar user interface as this program. Actually that program's user interface was much more complicated. The q and ? commands are exactly the same in the two programs. The r command from Lab 4 has the same format as the i, d, c, and r commands in this program. The d command from Lab 4 has the same format as the remaining commands for this program. Feel free to modify the given user interface code from Lab 4 to make it work here.

Command Line Argument: Debug Mode

Your program is to be able to take one optional command line argument, the -d flag. When this flag is given, your program is to run in "debug" mode. When in this mode, your program is to display each value and the result of the hash function on that value for the i, d and c commands. Assuming the table size is 8, the inserting of the value of 16 should have a debug message something like:

```
The value of 16 has a hash value of 0
```

If the table size was 11, inserting the value of 16 should have a debug message something like:

```
The value of 16 has a hash value of 5
```

When the flag is not given, this debugging information should not be displayed. One simple way to set up a "debugging" mode is to use a boolean variable which is set to true when debugging mode is turned on but false otherwise. This variable may be a global variable. Then using a simple if statement controls whether information should be output or not.

```
if ( debugMode == TRUE )
    printf (" Debugging Information \n");
```

Optional ways for writing this code can be seen at:

- https://www.cs.uic.edu/pub/CS211/LectureNotesF12/debugmode.c
- https://www.cs.uic.edu/pub/CS211/LectureNotesF12/debugmacro.c

Redirection of Input

To help test your program, the use of redirection of standard input from a text file is a good idea for this project. Redirection is done at the command line using the less than and greater than signs. Redirection of both input and output can be done; however, for this project, you may only want to use redirection of input. We talked about this in project 3.

- Assume you have a text file that is properly formatted to contain the input as someone would type it in for the input called: lab8input.txt
- Assume the executable for this project is in a file called: a.exe
- To run the project so that it reads the input from this text file instead of standard input using redirection of input, you would type the following on command line:

```
./a.exe < lab8input.txt
```

Program Submission

You are to submit the programs for this lab via the Assignments Page in Blackboard.

To help the TA, name your file with your net-id and the assignment name, like:

- ptroy1LabX.c