

CS271 Homework 5

Posted Online: Thursday 2007.04.12
Due Date/Time: Tuesday 2007.04.24 at Midnight
Description: Complete a C++ program that implements a Hash Table
Files to Submit: Makefile, main.cpp, node.h, hashtable.h, hashtable.cpp
Purpose: To understand and implement a simple Hash Table
Knowledge Needed: Hash Tables, Linked Lists
Homework Details: This is a simple version of a Hash Table.

- It stores C++ strings (string).
- The hash function can use the "modulo" method on the sum of all the ASCII values in the string.
- The hash table should use a linked list (LIFO) for collision resolution.
- The user gets to choose how big the hash table is by passing 1 command line parameter
- The user gets to add numbers to the hash table, and it will tell us whether it fit or if there was a collision, and what the hash function value was.
- The user gets to search for a number and get told whether it was found or not, and if found, how many nodes it had to look at before it found it (including the node it found) or didn't find it.

Hints:

- use argc & argv to extra the hash table size from the user
 - use the "atoi" ... ascii to integer function ... to convert the argv[0] string to an integer
Example: `#include <cstdlib>`
`int size = atoi(argv[0]) ;`
- Your hash table default constructor probably just some prime number (like 11) as default size
- Your hash table class constructor should probably take an integer size as the parameter
- Your hash table class should contain an array of linked lists.
- If the linked list is empty, that means that the address is open
- If the linked list has items in it, that means the address is used, and a collision will occur.
- You will need a node.h file because of the linked lists you're using
- You will need something like a public Insert function for the User
 - A Hash Table insert will always be successful (unless the program runs out of memory)
 - The Insert function will need to somehow return whether it was a hit or collision
 - The Insert function will need to somehow return which hash address was used
 - Remember the insert is LIFO, so you simply insert a new head pointer each time
- You will need something like a public Search function for the User
 - A Hash Table search could be successful or unsuccessful depending on if it's found or not
 - The Search function will also need to somehow return how many nodes it had to look at before it
- You will need a private Hash function which is called by both the Insert and Search functions
 - This is the function that sums up the ASCII values of all the characters in the string and uses the modulo of the hash table size to find where to put it.
- You may want a Hash Table Print function for debugging purposes
 - You could even add this Print function as a menu option if you like although not required
 - This would simply loop through the hash table and print each linked list on a new line
- You will need a main program file that has a menu, creates a Hash Table, and calls the appropriate member functions based on the menu options chosen

~> ./homework5 10

Hash Table Size chosen was 10.

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): 1

Input a String: **ABC**

Hash Address Used: 8

// $65+66+67 = 198 \% 10 = 8$

No Collision

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): 1

Input a String: **Dd**

Hash Address Used: 8

// $68 + 100 = 168 \% 10 = 8$

Collision Occurred

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): 1

Input a String: **vP**

Hash Address Used: 8

// $118 + 80 = 198 \% 10 = 8$

Collision Occurred

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): 2

Input a String: **vP**

1 Nodes Searched
Record Found!

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): **2**
Input a String: **ABC**
3 Nodes Searched
Record Found!

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): **2**
Input a String: **XYZ**
0 Nodes Searched
Record Not Found!

// $88 + 89 + 90 = 267 \% 10 = 7$

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): **2**
Input a String: **X**
3 Nodes Searched
Record Not Found!

// $88 = 88 \% 10 = 8$

Homework 5 MAIN MENU

1 – Insert integer into Hash Table
2 – Search for integer in Hash Table
3 – Print Hash Table
X - EXIT PROGRAM

// note #3 is optional, not required for homework 5

Choose an Option (hit enter): **X**