**Data Structures (CS3424) – Section 1**

**Fall 2013**

**Lab 10: Associative Containers (Sets/Multisets, Maps/Multimaps, and Hashing)**

**Name**: James Williams

**Grading**: Each problem – 2 points, the programming part – 4 points (Total: 20 points)

**Objectives**: (1) Compare and contrast the associative containers (Sets/Maps/Hashing).

Part I: Sets/Multisets

1. Problem #1 (page 520 in the textbook)  
   cout << t << endl; // print the string “123”  
   cout << (t-s) << endl; // prints the string “123”  
   cout << (s-t) << endl; // prints the string set {“hello”, “bye”}  
   cout << contains(s, “ace”) << endl; // returns 0, because “ace” is not in set s  
   cout << contains(s, “123”) << endl; // returns 0, because “123” is not in set s  
   s = s \* t; // creates the intersecting set s from the contents of s and t  
   cout << s << endl; // prints the set s { }, because s and t had no common values  
   t = t \* s; // creates an empty set t, because set t and s share no common vales
2. Problem #4 (page 520)  
   m = {25, 37, 27, 37}  
   n = {25, 37, 27}  
   m = {27}  
   n = {27}

Part II: Maps/Multimaps

1. Problem #1 (page 530)  
   a) ISBN the number is never re-used for any other book  
   b) I would use the name, because the players name has more characters to help it remain unique. In the case it is not unique, you could then add a integer to the end of it, while having it mapped to a uniform number. The combination of those choices should statistically hold a high probability of remaining a many to one key, value map and never a many to many or one to many relationship.  
   c) Model number most definitely, because companies like to serialize the model numbers to be unique to their brand.   
   d) Course Title is generally a good serialization method, but there can be multiple sections for a single course title. That being said, it seems like the best method (depending on how the information is structured) would be to use the course ID.
2. Problem #2 (page 530)  
   a\_map key ---- a\_map values  
   ~~1234 ---- “sam”~~2456 ---- “bill”  
   2222 ---- “hal”  
   1234 ---- “sally”   
   cout << a\_map[1111]; // should display / print no value or null, because it’s value has not been set yet.   
   Using an iterator to print the keys and values would print out as follows:  
   1234:”sally”, 2456:”bill”, 2222:”hal”, 1111:null
3. Problem #3 (page 530)  
   1234: “sam”, “sally”  
   2456: “bill”  
   2222: “hal”  
   1234: “sally”  
   1111: null

Part III: Hashing

1. Problem #2 (page 541/542)  
     
   Linear probing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Hash\_fcn() | Hash\_fcn()%table\_size | Hash\_fcn()%7 | Hash\_fcn()%13 |
| “Tom” | 84274 | 4 | 1 | 8 |
| “Dick” | 2129869 | 4 => 0 | 0 | 1 |
| “Harry” | 69496448 | 3 | 0 => 2 | 8 => 9 |
| “Sam” | 82879 | 4 => 1 | 6 | 4 |
| “Pete” | 2484038 | 3 => 2 | 4 | 11 |

Chaining

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Hash\_fcn() | Hash\_fcn()%table\_size | Hash\_fcn()%7 | Hash\_fcn()%13 |
| “Tom” | 84274 | 4 => [0] | 1 => [0] | 8 => [0] |
| “Dick” | 2129869 | 4 => [2] | 0 =>[0] | 1 => [0] |
| “Harry” | 69496448 | 3 => [0] | 0 => [1] | 8 => [1] |
| “Sam” | 82879 | 4 => [1] | 6 =>[0] | 4 => [0] |
| “Pete” | 2484038 | 3 => [1] | 4 => [0] | 11 => [0] |

1. Problem #3 (page 541/542)  
   The following table displays the placement of key values using c:2 (linear probing), and c:3 (chaining)

|  |  |  |
| --- | --- | --- |
| Index | Key | Chaining |
| 0 | 24 – placed due to collision | 20 |
| 1 | 6 – correct place | 6 |
| 2 | 20 – placed due to collisions |  |
| 3 | NULL |  |
| 4 | 14 – correct place | 14, 24 |

1. Problem #4 (page 541/542)  
   Deleting “Sam” can throw off the search for the other values (sam, pete, and harry), because their %5 value = 4 also. Since Sam was the first name to take the index at 4, the others will not be searchable unless the programmer knows to continue the search. If the value of NULL is set, then that could be an indicator that the bit was never used, but if a string of “deleted” or noted somehow that it contains a dirty/used bit, then the search could continue in efforts to finding the next value stored with a %5 = 4.
2. Programming #1 (page 541/542)  
   fairly confused by this question…   
     
   int findPosition(<vector> table, Key\_Type key){  
    int found = 0;   
    if(

}

**Turn In**: Hardcopy (during the class or submit to J-385), or softcopy (D2L).