CS 251, Spring 2009
Voicu Popescu
Assignment 7
Due Monday March 23rd, 11:59PM

Hash Tables

1. Implement a C++ hash table class **HashTable** in files HashTable.cc and HashTable.h
   a. The items stored are integers which serve both as keys and as elements.
   b. The hash table uses double hashing
      i. The primary hashing function is $h(k) = k \bmod N$, where k is the key, and N is a prime positive integer provided as input. Thus your table will have room for N entries.
      ii. Collisions are resolved using a secondary hashing function $h_2(k) = (h(k) + j * (q - k \bmod q)) \bmod N$, where j = 0, 1, …, N-1, and q is a prime positive integer provided as input.
   c. The constructor **HashTable**(int *N*, int *q*, char ***filename**) builds the hash table from keys stored in the file **filename**. The parameters *N* and *q* were defined above. The file is a text file with *n*+1 whitespace and/or new line delimited positive integers. The first integer is *n*, which gives the number of keys. The following *n* integers are the *n* keys.
   d. The void **Insert**(int *k*) method inserts the key *k* using the double hashing described above if the table is not full, and does not do anything if the table is full.
   e. The int **Find**(int *k*) method returns the index of a table entry that stores key *k*, if one exists, and -1 otherwise.
   f. The void **Remove**(int *k*) method makes available the first table entry that contains the key *k* (by order visited using the hash functions) and does not do anything if the table does not contain key *k*.
   g. The void **Print**(char ***filename**) method writes in a file called **filename** all table entries in ascending order of the table slot number, one per line. An entry that is available is printed as **index** "-1", where index is the entry's index in the table. An entry that stores a key is printed as **index key**. See the sample input/output files sampleinput_N=7_q=3.txt which uses N=7 and q=3 and sampleoutput_N=7_q=3.txt for input/output formatting.
   h. The destructor **~HashTable()** deallocates any used memory.
2. Number of probes.
   a. For N = 7,919 and n = 3,000 (q unspecified), what is the expected number of probes per insert, assuming the n numbers are randomly selected? Write your answer in a plain text file named "q2a.txt". When writing math, make it legible. For example, n(n+1)/2 and O(n^2) are ok. O(n2) isn't the same as O(n^2). A short justification is required.
   b. Measure the maximum and average number of probes per insert for pseudo-randomly generated numbers for N=7919, n=3000, and q = 1493. That is, run a test and report your results. Write your answer in a plain text file named "q2b.txt". You do not need to turn in source code for this part.
3. Extra-credit [4%] Given a set of n unit squares defined on a regular 2D grid, eliminate all internal edges (see Figure 1) in worst-case linear time (in n) using a hash table. The set of unit squares is dense, meaning that A is $O(n)$, where A is the area of the axis aligned bounding box of the squares. In a file ec.cc, create a program with a main function with the following functionality.
   a. A square is defined by the bottom left vertex (x_0, y_0); the other three vertices are (x_0+1, y_0), (x_0, y_0+1), and (x_0+1, y_0+1). All will be nonnegative numbers.
   b. An edge is a unit segment [$(x, y), (x, y+1)$] or [$(x, y), (x+1, y)$] that belongs to at least one unit square in the given set.
   c. An internal edge is an edge shared by 2 squares.

d. The program should take an input file name as the first parameter and an output file name as the second parameter on the command line. The input file is in text format and contains 1+2*n positive integers, delimited by whitespace and or new lines. The first number is the number of unit squares, the remaining numbers specify the bottom left vertex (in (x,y) pairs) for the n unit squares. Output the segments in any order. See the sample input and output files ecsampleinput.txt and ecsampleoutput.txt for formatting details. You should be able to exactly duplicate ecsampleoutput.txt (up to ordering of edges) given ecsampleinput.txt.
e. Your algorithm will receive no extra credit if it does not run in O(n).

4. Requirements
   a. Do not change the names of any of the files given.
   b. Do not include code in files other than those given.
   c. Do not change the signatures of any of the functions given.
   d. You may add functions and data as you need.
   e. Your code is REQUIRED to compile with /opt/csw/gcc3/bin/g++ (gcc version 3.4.5) on lore.cs.purdue.edu. If your code doesn't compile with this compiler, then even if it compiles with another, your code will still be considered noncompiling.
5. Grading
   a. grading will be primarily through testing the output of each of the required functions against the desired output in our own main program. Thus it is important that your output format match the expected output format.
6. Turn in instructions:
   a. Turn in files: HashTable.cc, HashTable.h, q2a.txt, q2b.txt, ec.cc (optional, do not turnin unless you have attempted the extra credit).
   b. Directory structure: all files in the directory A7 (the same directory that the assignment was distributed in). Other files in the directory will not be graded.
   c. in the directory above A7, run the command on lore.cs.purdue.edu
      - turnin -c cs251 -p A7 A7
   d. optionally run the following command to verify what files were submitted
      - turnin -c cs251 -p A7 -v
   e. Warning: turnin will be automatically disabled at 03/24/2009 at 12:00 AM EST and you will be unable to turnin your assignment after this time.
   f. Warning: turning in multiple times is ok but only the last one will be graded as each turnin permanently overwrites the previous one.
   g. Warning: failure to turnin exactly according to the instructions given above will result in your A7 receiving a grade of 0.

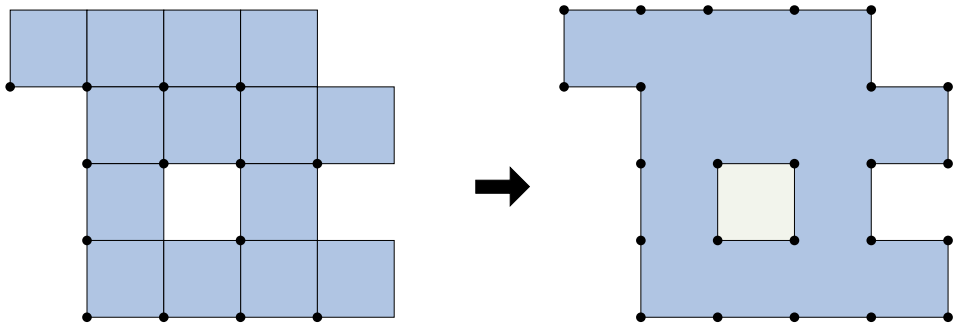

Figure 1. The set of 14 unit squares (*left*) is converted to the contour segments (*right*).