**Name:** _____

This assignment focuses on *hashing*, described in Chapter 20 of the textbook.[1] You should submit (1) a hardcopy of this assignment with your answers filled in neatly and (2) a well-packaged electronic submission with your answers to the programming components. (See below.)

All **electronic submissions** must be made using the Web interface at `http://cs.umaine.edu/~chaw/u/` only. Do not use email or any other means for submission. You should submit a single file named using the template `cos226-hw01-`*`lastname`*`-`*`firstname`*`-`*`pqrs`*`.tgz` where *`lastname`* and *`firstname`* are replaced by the obvious and *`pqrs`* is replaced by a 4-digit number of your choosing. If your submission is successful, you will be presented with a Web page indicating so, along with a timestamp and MD5 checksum; you should save that information for your records. You should design your submission so that the command `tar zxf cos226-hw01-`*`lastname`*`-`*`firstname`*`-`*`pqrs`*`.tgz` results in the creation of a directory `cos226-hw01-`*`lastname`*`-`*`firstname`*`-`*`pqrs`*. In that directory should be all the source code for your submission (organized in further sub-directories if you like) as well as a README file and a Makefile with the usual semantics. Do not submit any kind of non-source files (executables, object files, class files, etc.). Be especially careful if you use IDEs such as Eclipse for your work because the packages created by them often contain compiled, not source files.

*Ask for clarifications in class or on the newsgroup if you have any doubts regarding the submission format and procedure.*

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) Package your solutions to the programming questions as in previous assignments. Submit your work via `http://cs.umaine.edu/~chaw/u/`. *After* submitting your work, complete the following:
   File name: _____ Size, in bytes: _____
   MD5 checksum: _____
   Timestamp: _____

3. (6 pts.) Compare search trees and hashing as data structures for collections of keys by naming (1) three operations that are supported by both and (2) three operations that are supported by search trees, but not by hashing.

[1] Mark Allen Weiss, *Data Structures and Problem Solving Using Java*, 4th edition (Addison-Wesley, 2010).

4. (6 pts.) Refer to the simple method for storing a set of 16-bit integers described at the beginning of Section 20.1. How much storage (in bytes) would be required if that method were used for 64-bit integers? Explain how you arrive at your answer.

5. (6 pts.) The book notes that a hash table may be used to implement a set in constant time per operation. Name three set operations that a hash-table may implement in constant time. Name three set operations that a hash table cannot easily implement in constant time. Justify your answers.

6. (6 pts.) What is the maximum number of collisions that may result from hashing $n$ keys? Provide an explicit hash function that produces this number of collisions.

7. (6 pts.) If a hash table $H$ of size $m$ contains $n$ keys, what are the minimum and maximum number of probes required to insert a new key into $H$? Justify your answer.

8. (8 pts.) Using the method of the fourth paragraph of Section 20.1, map each of the 10 space-separated strings `The vorpal blade went snicker-snack!` `He went galumphing back.` (including the hyphen, exclamation point, and period) to integers. You may use a calculator or computer *but must show your work for at least the string* ***galumphing***. Assume a 7-bit ASCII character encoding; for convenience, a table of codes appears below. Note that there can be no partial credit awarded if you arrive at a wrong answer and do not show your work.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | 16 | DLE | 32 | | 48 | 0 | 64 | @ | 80 | P | 96 | ‘ | 112 | p |
| 1 | SOH | 17 | DC1 | 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 2 | STX | 18 | DC2 | 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 3 | ETX | 19 | DC3 | 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 4 | EOT | 20 | DC4 | 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 5 | ENQ | 21 | NAK | 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 6 | ACK | 22 | SYN | 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 7 | BEL | 23 | ETB | 39 | ’ | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 8 | BS | 24 | CAN | 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 9 | HT | 25 | EM | 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 10 | LF | 26 | SUB | 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 11 | VT | 27 | ESC | 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 12 | FF | 28 | FS | 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | \| |
| 13 | CR | 29 | GS | 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 14 | SO | 30 | RS | 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 15 | SI | 31 | US | 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | DEL |

9. (8 pts.) Repeat Question 8 using the hash function of Figure 20.1 with `tableSize` = 17. As in that question, show your work at least for the string `galumphing`, here by noting the value of `hashVal` after each iteration of the for loop.

10. (8 pts.) Repeat Question 9 for the hash function of Figure 20.2. Justify the textbook's claim "... the result obtained by allowing overflow... not the same... every step" by providing a concrete example of different results.

11. (8 pts.) Repeat Question 10 for the hash function of Figure 20.3.

12. (4 pts.) Suppose we are writing code that must run in a programming environment that supports only addition and shifting of 16-bit binary numbers (no multiplication or other arithmetic operations). Provide efficient, detailed pseudocode for this environment that is equivalent to line 12 of Figure 20.2 in the textbook.

13. (8 pts.) Using Figure 20.5 in the textbook as a template, depict the result of inserting the strings of Question 8 into a hash table of size 17, using each of the three hash functions noted in Questions 9, 10, and 11. Use the linear probing method for collision resolution. It may be convenient to depict the tables in landscape mode on this page.

14. (8 pts.) Repeat Question 13 using the quadratic probing method for collision resolution.

15. (8 pts.) Repeat Question 13 using the separate chaining method for collision resolution, and a hash table of size 11.

16. (8 pts.) Consider the following modification of the separate chaining method, yielding a method called *coalesced hashing*: Instead of using separate storage for the linked lists used by separate chaining, the linked lists are stored in the otherwise unused slots of the hash table. With this method, each slot in the hash table contains either a key (as in linear or quadratic probing) or a cell belonging to a linked list that is used for chaining (as in separate chaining). Note that the linked-list cell occupying a hash table slot does not necessarily contain a key that hashes to that slot.

    Repeat Question 13 using this coalesced chaining method for collision resolution.

17. (100 pts.) Modify the textbook's implementation of the `HashSet` class[2] to use the *coalesced hashing* technique of Question 16 instead of quadratic probing. You may use as much or as little of the textbook's code as you wish (with proper attribution, as always) but your implementation of `HashSet` must be *completely interchangeable* with the textbook's implementation. That is, any program that uses the textbook's implementation should work as before if all the files implementing the textbook's HashSet class are replaced with those implementing yours (with no other changes).

[2] *Idem*, §20.4.1.