You will submit your solution to this assignment to the Curator System (as `HW4`). Your solution must be either a plain text file (e.g., NotePad) or a MS Word document; submissions in other formats will not be graded.

Except as noted, credit will only be given if you show relevant work.

---

1. Consider a hash table consisting of $M = 11$ slots, and suppose nonnegative integer key values are hashed into the table using the hash function `h1()`:

```
int h1 (int key) {
  int x = (key + 5) * (key + 5);
  x = x / 16;
  x = x + key;
  x = x % 11;
  return x;
}
```

The key values are the same for all three parts, so we might as well compute the hash function values for all 10 of them:

h1(43) = 0, h1(23) = 5, h1(1) = 3, h1(0) = 1, h1(15) = 7, h1(31) = 2, h1(4) = 9, h1(7) = 5, h1(11) = 5, h1(3) = 7

a) [20 points] Suppose that collisions are resolved by using linear probing. The integer key values listed below are to be inserted, in the order given. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (if any) for each key, and the final contents of the hash table after the following key values have been inserted in the given order:

Key Value	Home Slot	Probe Sequence
43	**0**	
23	**6**	
1	**3**	
0	**1**	
15	**7**	
31	**2**	
4	**9**	
7	**5**	
11	**5**	**6, 7, 8**
3	**7**	**8, 9, 10**

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	**43**	**0**	**31**	**1**		**7**	**23**	**15**	**11**	**4**	**3**

b) [20 points] Suppose that collisions are resolved by using quadratic probing, with the probe function:

$$\left(k^2 + k\right)/2$$

The integer key values listed below are to be inserted, in the order given. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (if any) for each key, and the final contents of the hash table after the following key values have been inserted in the given order:

Key Value	Home Slot	Probe Sequence
43	**0**	
23	**6**	
1	**3**	
0	**1**	
15	**7**	
31	**2**	
4	**9**	
7	**5**	
11	**5**	6, 8
3	**7**	8, 10

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	**43**	**0**	**31**	**1**		**7**	**23**	**15**	**11**	**4**	**3**

c) [20 points] Suppose that collisions are resolved by using double hashing (see the course notes), with the secondary hash function `Reverse(key)`, which reverses the digits of the key and returns that value; for example, `Reverse(7823) = 3287`.

The integer key values listed below are to be inserted, in the order given. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (if any) for each key, and the final contents of the hash table after the following key values have been inserted in the given order:

Key Value	Home Slot	Probe Sequence
43	**0**	
23	**6**	
1	**3**	
0	**1**	
15	**7**	
31	**2**	
4	**9**	
7	**5**	
11	**5**	**Reverse(11) = 11, so sticks on slot 5, fails**
3	**7**	**Reverse(3) = 3, so: 10**

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	**43**	**0**	**31**	**1**		**7**	**23**	**15**		**4**	**3**

2. [20 points] A pseudo-random generator is a software function that, when called N times, returns a sequence of N integer values $\{V_1, V_2, V_3, \ldots, V_N\}$ that have certain nice statistical properties, like being uniformly distributed over some range. A pseudo-random generator is "primed" by supplying an initialization value called a *seed*. Different seed values result in different sequences of values, but for a given seed value the resulting sequence of values is always the same.

   Could a pseudo-random generator `rand()` could be used as part of a probing strategy to resolve collisions in a hash table? If no, explain why not? If so, describe an effective solution that would yield different probe sequences for different key values that collide in the same home slot.

   The first issue is reproducibility. We must be able to generate the same sequence of slots whenever we probe, whether it's during insertion or search or deletion.

   That's easy enough to solve; all we need to do is consistently seed the random generator with the same value when we are dealing with the same key value.

   The second issue is to get different probe sequences with different keys values that collide on the same home slot. There may be several options…

   If the key values are integers, just use the key values themselves to seed the random generator. If the seed values are not integers, even though they collide in the same home slot, the hash function may still yield different integers from each key (the collision may be due to modding by the table size). In that case, we could seed with the hash function's value (before modding). If the hash function does map two different keys to the same integer value, we could still use the hash function's value, but add a counter for the number of keys that have collided in the slot; of course, we'd need to store that counter value for each key so we could use it again whenever we search for that key.

3. [20 points] Most programming languages that have a `switch` statement limit the variable used to specify the switch target to be of some integer type. BTW, the point of a `switch` statement is that the cost of jumping to the correct case is Theta(1).

   However, Java 7 adds support for `switch` statements that use string variables to specify the switch target. Explain clearly, how the Java compiler can accomplish this.

   The simplest way to think about the issue is that the Java 7 compiler must transform a switch statement using strings for the switch variable and the labels into one that uses integers for the switch variable and the labels.

   The compiler could use a hash function to map the actual label strings to integers, and create a new integer variable to use for the switch.

   But if the hash function isn't 1-1, this would yield a switch with duplicate labels…

   But, the compiler will get to see all the actual labels when it parses the code, so if the compiler generates a perfect hash function for those strings then everything would work correctly. At runtime, the actual string would be hashed to an integer and that would be mapped to a case label.