1. Fill in the proof of correctness for the method `strToInt` on the next page. It returns the `int` value corresponding to the decimal number written in the first `n` characters of the array `s`.

   That code references a function `charToInt` that takes a character in the range `‘0’`, `‘1’`, …, `‘9’` to the corresponding `int` in the range `0`, `1`…, `9`. It has the following implementation:

   ```
   // Precondition: ‘0’ <= ch <= ‘9’
   int charToInt(char ch) {
     return ch – ‘0’;
   }
   ```

   Reason in the direction (forward or backward) indicated by the arrows on each line: forward outside the loop and backward inside the loop.

   In addition to filling in each blank below, you must provide additional explanation whenever two assertions appear right next to each other, with no code in between: in those cases, explain why the top statement *implies* the bottom one. You can skip this explanation if the two statements are identical or if the bottom one simply drops facts included the top one.

   Notes on the notation used:

   - A summation over a range like “s[a] + … + s[b]” should be interpreted as 0 if there are no indexes between the lower bound, a, and the upper bound, b, (i.e., if b < a).
   - The assertions make reference to a *mathematical* function “int” that takes a character in the range ‘0’, ‘1’, …, ‘9’ to the corresponding integer value in the range 0, 1, .., 9. (The Java function `charToInt` mentioned above implements this function.)

{{ Precondition: 0 < n <= s.length() }}

`int strToInt(char[] s, int n) {`

↓ `int i = 0;`

{{ Precondition and i = 0 }}

↓ `int val = 0;`

{{ Precondition and i = 0 and val = 0 }}

Since i = 0, there are no indices in 10^{i-1} * int(s[0]) + … + 10 * int(s[i-2]) + int(s[i-1]), which means that this sum is 0 = val.

{{ Inv: val = 10^{i-1} * int(s[0]) + … + 10 * int(s[i-2]) + int(s[i-1]) }}

`while (i != n) {`

Multiplying both sides by 10 and adding int(s[i]) gives what we need below.

{{ 10 * val + int(s[i]) = 10^{i} * int(s[0]) + … + 10 * int(s[i-1]) + int(s[i]) }}

↑ `int d = charToInt(s[i]);` // in our notation, now d = int(s[i])

{{ 10 * val + d = 10^{i} * int(s[0]) + … + 10 * int(s[i-1]) + int(s[i]) }}

↑ `val = 10 * val + d;`

{{ val = 10^{i} * int(s[0]) + … + 10 * int(s[i-1]) + int(s[i]) }}

↑ `i = i + 1;`

{{ val = 10^{i-1} * int(s[0]) + … + 10 * int(s[i-2]) + int(s[i-1]) }}

↑ `}`

↓

{{ val = 10^{i-1} * int(s[0]) + … + 10 * int(s[i-2]) + int(s[i-1]) and i = n }}

Since i = n, we can substitute it into the val = … part , giving the claim below.

{{ Postcondition: val = 10^{n-1} * int(s[0]) + … + 10 * int(s[n-2]) + int(s[n-1]) }}

`return val;`

`}`