CSE 331 Spring 2020 Homework 2 (part 1)

1. Fill in the proof of correctness for the method on the next page, `strToInt`, which takes a string representation of a decimal number and returns the `int` value that it represents.

   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:

   - In an expression like "10 * s[i-1] + s[i]", the characters s[i] and s[i-1] should be interpreted as digits. For example, if s[i-1] = '3' and s[i] = '5', then this expression would mean 10 * 3 + 5 = 35. (In Java, you must convert a '0'-'9' character into the equivalent int value, but we will ignore that distinction where it seems clear.)
   - A summation over a range like "s[0] + … + s[i-1]" should be interpreted as 0 if there are no indexes in the range, i.e., if i = 0.

{{ Precondition: s.length > 0 }}

```
int strToInt(String s) {
  int n = s.length();
```
{{ _____ }}
```
  int i = 0;
```
{{ _____ }}
```
  int val = 0;
```
{{ _____ }}

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

```
  while (i != n) {
```

{{ _____ }}
```
    int d = s.charAt(i) – '0';          // in our notation, now d = s[i]
```
{{ _____ }}
```
    val = 10 * val + d;
```
{{ _____ }}
```
    i = i + 1;
```
{{ _____ }}
```
  }
```

{{ _____ }}

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

```
  return val;
}
```

2. In this problem, parts (a-c), you will implement the same method from Problem 1, `strToInt`, but in three slightly different ways.

   In each part, some of the code for that implementation is provided. You must fill in the missing code so that the resulting method body correctly implements `strToInt`.

   In each part, the **loop invariant** is also provided. Your code must be correct using the provided loop invariant. Code that correctly implements the method but does not satisfy the invariant will lose points.

   The precondition and the postcondition for all parts are exactly the same as in Problem 1. (I.e., each part really is asking you to implement the same method as before.)

a)
```
{{ Precondition: s.length > 0 }}
int n = s.length();
int i = 0;
int val =


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




  i = i + 1;
}


{{ val = 10^(n-1) s[0] + … + 10 s[n-2] + s[n-1] }}
return val;
```

b) {{ Precondition: s.length > 0 }}

```
int n = s.length();
int i = -1;
int val =


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




}


{{ val = 10^(n-1) s[0] + … + 10 s[n-2] + s[n-1] }}
return val;
```

c) {{ Precondition: s.length > 0 }}

```
int n = s.length();
int i = 1;
int val =


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




}


{{ val = 10^(n-1) s[0] + … + 10 s[n-2] + s[n-1] }}
return val;
```