CSE 331 Summer 2021 HW2

General Rules:

- For logical operators, you may use words (e.g., “or”) or any standard symbols (e.g., “V”).
- Assume that
    - all numbers are integers
    - integer overflow will never occur
    - integer division rounds toward zero (as in Java) (Example: 5 / 2 = 2)
- Simplify but do not weaken your assertions.

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’;
}
```

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.)

Reason in the direction (forward or backward) indicated by “(fwd)” and “(bck)” on each line (forward and backward, respectively): 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 in the top one. (Each triple that requires additional explanation is marked with a “ ? “ on the left.)

{{ Precondition: 0 < n <= s.length() }}
int strToInt(char[] s, int n) {

int i = 0;

(fwd) {{ _____ }}

int val = 0;

(fwd) {{ _____ }}

?

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

while (i != n) {

?

(bck) {{ _____ }}

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

(bck) {{ _____ }}

val = 10 * val + d;

(bck) {{ _____ }}

i = i + 1;

(bck) {{ _____ }}

}

(fwd) {{ _____ }}

?

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

return val;

}

2 - Fill in the missing parts of each implementation of the method *removeChar* below. It takes as input a string s and a character c, and it returns s with all the c's removed. In each case, the precondition and postcondition are the same, so each loop is trying to accomplish the same task, but the provided code or the invariant is slightly changed, so the details of how the code will accomplish it should be different.

In each case, your code must be correct according to the loop invariant that is provided. You do not need to turn in your proof of correctness. However, since your code will be graded on its correctness, you should still complete this for yourself. Additionally, you may not

- add any additional statements outside of the loop,
- add any additional loops, or
- call any methods other than String.charAt, String.length, and StringBuilder.append.

Notes on the notation used:

- To formally argue correctness, we need a formal definition of what the code is supposed to do. The function "del_c" referenced in the assertions below does that. Informally, del_c(s) means the string s but with all c's removed.
- Formally, we can define del_c recursively (as in CSE 311) as follows:
    - del_c("")= ""
    - del_c(wa) = del_c(w) if a = c
    - del_c(wa)= del_c(w)a otherwise

      (Here, "w"represents any string and "a"any single character. Each string can either be written in the form wa for some w and some a or it is the empty string "".)
- The notation "s[i.. j]" means the substring from index i up to and including index j. For example, if s = "abcde", then s[2.. 3] = "cd". If j = i–1, then this indicates an empty string. In symbols, we have s[i.. i-1] = "".

a)
```
{{ Precondition: s != null }}
String removeChar(String s, char c) {
  StringBuilder t = new StringBuilder();
  int i = ______


  {{ Inv: t = del_c(s[0 .. i-1]) }}
  while ( _________________ ) {


    i = i + 1;
  }
  {{ Postcondition: t = del_c(s) }}
 return t.toString();
}
```

b)
```
{{ Precondition: s != null }}
String removeChar(String s, char c) {
  StringBuilder t = new StringBuilder();
  int i = ______

  {{ Inv: t = del_c(s[0 .. i]) }}
  while ( _________________ ) {


    i = i + 1;
  }
 {{ Postcondition: t = del_c(s) }}
 return t.toString();
}
```

c)
```
{{ Precondition: s != null }}
String removeChar(String s, char c) {
  StringBuilder t = new StringBuilder();
  int i = _______

  {{ Inv: t = del_c(s[0 .. i]) }}
  while ( ___________________ ) {
    i = i + 1;




  }
  {{ Postcondition: t = del_c(s) }}
  return t.toString();
}
```

d)
```
{{ Precondition: s != null }}
String removeChar(String s, char c) {
  StringBuilder t = new StringBuilder();
  int i = _______

  {{ Inv: t = del_c(s[0 .. i-1]) }}
  while ( ___________________ ) {
    i = i + 1;




  }
  {{ Postcondition: t = del_c(s) }}
  return t.toString();
}
```

3 - Fill in the missing parts of the implementation of the method movingAverage on the next page. It takes as input an array A and an integer k, along with an array B into which the output will be written. The goal is to write into B the average of each length-k subarray of A. I.e., B[0] = (A[0] + … + A[k-1]) / k and B[1] = (A[1] + … + A[k]) / k and so on.

Your code must be correct according to the loop invariant that is **provided**. You do not need to turn in your proof of correctness. However, since your code will be graded on its correctness, you should still complete this for yourself.

Additionally, **you may not** (1) add any additional loops or (2) call any other methods.

Notes on the notation used:

- Use “n” to denote the length of A. (The length of B will hence be n – k + 1.)
- Use “avg” to denote the function that averages the values in an array. In other words, “avg A[i .. j]” means (A[i] + A[i+1] + … + A[j]) / (j – i + 1).

```
{{ Precondition: 0 < k <= n = A.length and n – k + 1 = B.length }}
void movingAverage(float[] A, int k, float[] B) {
  float s = 0;
  int j = _______

  {{ Inv: s = A[0] + … + A[j-1] }}
  while ( ___________________ ) {




    j = j + 1;
  }

  int i = 0;



  {{ Inv: B[0] = avg A[0 .. k-1], B[1] = avg A[1 .. k], …, B[i] = avg A[i .. i+k-1] and
          s = A[i] + … + A[i+k-1] }}
  while ( ___________________ ) {




    i = i + 1;
  }

  {{ Postcondition: B[0] = avg A[0 .. k-1], B[1] = avg A[1 .. k], …, B[n-k] = avg A[n-k .. n-1] }}
}
```

4 - Fill in the missing parts of the method runLengthDecode on the next page. It takes as input an array of characters A and an array of integers B. The goal is to return a string that has B[0] copies of character A[0] followed by B[1] copies of character A[1] and so on.

In this case, the invariant for the outer loop is provided, but you must fill in the invariant for the inner loop.

Your code must be correct according to the loop invariants in your answer. You do not need to turn in your proof of correctness. However, since your code will be graded on its correctness, you should still complete this for yourself.

Additionally, **you may not** call any method other than StringBuilder.append. You can add additional **while** loops, but the invariant for each loop must also be given.

Notes on the notation used:

- Use “n” to denote the length of A.
- The invariant and postcondition use the notation “a * b”, when a is a character and b is a non-negative integer, to mean the string that has b copies of the character a. E.g., ‘w’ * 3 would be “www”.

```
{{ Precondition: 0 < n = A.length = B.length }}
String runLengthCode(char[] A, int[] B) {
  StringBuilder s = new StringBuilder();

  int i = _____________



  {{ Inv: s = A[0] * B[0] + A[1] * B[1] + … + A[i-1]  * B[i-1] }}
  while ( ___________________ ) {









  }



  {{ s = A[0] * B[0] + A[1] * B[1] + … + A[n-1] *  B[n-1] }}
  return s.toString();
}
```