CSC 240: Computer Science III
Fall 2026 – Section 01
Homework 1 – Text Processing (Chapter 9)

Name: _____

WCU ID: _____

Assigned: Thursday, September 3, 2026
Due: Tuesday, Sep 15, 2026, 11:59 PM

This homework contains 6 pages and 3 questions, worth a total of 100 points. It counts for **6%** of your course grade. Every question uses the Chapter 9 tools covered in class: the `Character` class, `String` methods, `StringBuilder`, tokenizing with `split` / `StringTokenizer`, and the numeric wrapper classes, e.g. `Integer.parseInt` and `Double.parseDouble`.

(**DO NOT copy code from the book, slides, StackOverflow, YouTube, GitHub, ChatGPT, etc.** See Section 6 of the syllabus.)

Instructions – read before you start

- **Individual work.** You may discuss ideas, but you may not share, exchange, or jointly write code. You must be able to explain and modify every line you submit.
- **Starter code.** Download `CSC240_HW1_starter.zip` from the course website (wcu.ninja/csc240) or D2L. It contains `TextAnalyzer.java`, `Word2Number.java`, `GasPrices.java`, and `gas_prices.csv`. **Do not change any of the existing code** in the templates; add your code where the `TODO` comments are. You may add helper methods.
- **Submission.** Put your name in the comment header of each file. Submit the three `.java` files in one zip named `LastName_FirstName_HW1.zip` to the D2L dropbox before the deadline. The D2L timestamp is authoritative.
- **Code must compile** with the JDK version listed in D2L. A file that does not compile receives at most half of that question's points. Partially working programs receive partial credit when you clearly comment what works and what does not.
- **Answers without code earn 0 points.** Every number or string your program prints must be computed by your Java code, not typed in by hand.
- **Style** (up to −5 points overall): meaningful variable names, consistent indentation, and a short comment for each part (a), (b), ...
- **Late work:** no credit unless a valid excuse is communicated before the deadline (syllabus, Section 7).

Question:	1	2	3	Total
Points:	30	25	45	100
Score:				

1. **Text Analyzer (30 points)** – `Character` class, `String` methods, `StringBuilder`

   Complete `TextAnalyzer.java`. The program reads **one line of text** from the user with `Scanner.nextLine()` and then prints the following report. Print the labels exactly as shown in the sample runs.

   (a) (8 points) **Character classes.** Count how many characters in the line are *letters*, *digits*, *whitespace*, and *other* (everything else, e.g. punctuation). Use `charAt`, `Character.isLetter`, `Character.isDigit`, and `Character.isWhitespace`. The four counts must add up to `line.length()`.

   (b) (6 points) **Vowels.** Count the vowels (`a e i o u`). The count must be case-insensitive; use `Character.toLowerCase` (or `toUpperCase`) rather than listing ten letters.

   (c) (6 points) **Words.** A *word* is a run of non-whitespace characters. Print the number of words and the *longest* word (if several words tie for longest, print the **first** one). Hint: `line.trim().split("\\s+")` or a `StringTokenizer`. An empty line has 0 words.

   (d) (5 points) **Reversed.** Print the whole line reversed. You must use `StringBuilder` (its `reverse()` method is fine).

   (e) (5 points) **Palindrome.** Print `true` if the line reads the same forwards and backwards when you **ignore case and ignore every character that is not a letter or a digit**; otherwise print `false`. Suggested approach: build a "cleaned" `StringBuilder` containing only `Character.isLetterOrDigit` characters converted to lower case, then compare it with its reverse using `equals`.

Sample run 1

```
Enter a line of text: Java is fun, but StringBuilder is faster! 2026
Letters:    33
Digits:     4
Whitespace: 7
Other:      2
Vowels:     12
Words:      8
Longest:    StringBuilder
Reversed:   6202 !retsaf si redliuBgnirtS tub ,nuf si avaJ
Palindrome: false
```

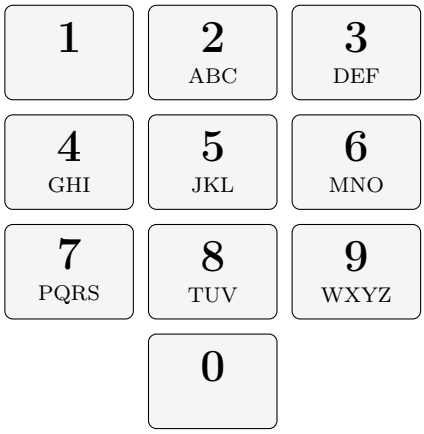
Sample run 2

```
Enter a line of text: A man, a plan, a canal: Panama
Letters:    21
Digits:     0
Whitespace: 6
Other:      3
Vowels:     10
Words:      7
Longest:    canal:
Reversed:   amanaP :lanac a ,nalp a ,nam A
Palindrome: true
```

(In sample run 2, `canal:` and `Panama` both have 6 characters; `canal:` comes first.)

2. (25 points) **Word to Number** – `Character` conversion, `StringBuilder`

   Phonewords are telephone numbers written with letters, such as 1-800-Flowers or 1-800-Contacts. On a telephone keypad each digit from 2 to 9 is labeled with letters, so a phoneword can be converted back to an all-numeric number:



   Complete `Word2Number.java`. The template reads one line from the user. Convert it so that

   - every **letter**, upper *or* lower case, is replaced by its keypad digit (`A`, `B`, `C`, `a`, `b`, `c` → `2`, and so on);
   - every other character (digits, dashes, spaces, parentheses, ...) is copied **unchanged**.

   Requirements.

   - Build the result in a `StringBuilder`; do not build it with repeated string concatenation (`result = result + c`) inside the loop.

- Write a helper method `public static char letterToDigit(char letter)` that returns the digit for one letter. Use `Character.toUpperCase` (or `toLowerCase`) so the method handles both cases without listing 52 characters. A `switch`, a chain of `if`s on character ranges (`c >= 'A' && c <= 'C'`), or a lookup `String` are all acceptable.
- Do not use regular expressions or `String.replace` for the conversion; the point is to process the text one character at a time.

Examples

```
Input a number: 1-800-Flowers
1-800-3569377
```

```
Input a number: 1-800-Contacts
1-800-26682287
```

```
Input a number: 1-800-got-junk
1-800-468-5865
```

```
Input a number: (610) 436-CSC-240
(610) 436-272-240
```

3. **Gas Prices (45 points)** – tokenizing a CSV file, wrapper-class parsing, `String` methods

   `GasPrices.java` downloads a CSV file from the course website and loads it into a String[] array named gasData, one line per element. (If the download fails on your computer, the template automatically reads the local copy `gas_prices.csv` instead; keep that file in the folder you run the program from. **Do not modify `getData`.**)

   The file holds the **weekly average price of a gallon of regular gasoline** for New York State and eight of its regions from January 2008 through December 2018. The first few elements of gasData look like this:

```
Date,New York State Average,Albany Average,Binghamton Average,Buffalo
Average,Nassau Average,New York City Average,Rochester Average,Syracuse
Average,Utica Average
12/31/18,2.64,2.51,2.55,2.7,2.58,2.7,2.61,2.52,2.62
12/24/18,2.69,2.56,2.61,2.75,2.63,2.75,2.66,2.57,2.67
12/17/18,2.72,2.59,2.65,2.8,2.67,2.78,2.69,2.6,2.71
12/10/18,2.76,2.62,2.69,2.86,2.7,2.81,2.74,2.64,2.75
and so on ...
```

   Important details about the data:

- gasData[0] is the **header row**. Skip it when you compute anything.
- Every data row has exactly 10 comma-separated fields, in this order:

> **Date, New York State Average, Albany Average, Binghamton Average, Buffalo Average, Nassau Average, New York City Average, Rochester Average, Syracuse Average, Utica Average**

- The date is written `M/D/YY`. The month and day are **not** zero-padded (`1/7/08` is January 7, 2008; `12/31/18` is December 31, 2018). The year is two digits; add 2000 to it. Hint: split the date on `"/"`, or use `indexOf('/')`, `lastIndexOf('/')`, and `substring`.
- Prices are text. Convert them with `Double.parseDouble` before doing arithmetic; convert month/day/year with `Integer.parseInt`.
- The rows are ordered from the newest date (top) to the oldest date (bottom).

Write Java code in `main` (plus any helper methods you like) that processes the array and prints the answers to the following. Print money with two decimals using `System.out.printf("%.2f", ...)`. Correct answers **without** code earn 0 points.

(a) (8 points) **Record count and date range.** Print how many weekly records the file contains (do not count the header) and the date range in the form `From <oldest date> to <newest date>`. *Self-check: there are 574 weekly records.*

(b) (12 points) **Average price per year for Buffalo.** For each year 2008 through 2018, print the average of the "Buffalo Average" column over all weeks of that year, one year per line. *Self-check: the 2008 average for Buffalo is $3.50.*

(c) (12 points) **Average price per calendar month for New York City.** For each of the 12 calendar months (January through December), print the average of the "New York City Average" column over **all weeks that fall in that month in any year** (so the "Jan" line averages every January week from 2008–2018). Print the month names, one per line. *Self-check: the January average for New York City is $3.03.*

(d) (13 points) **Highest and lowest price.** Find the highest and the lowest value in the "New York State Average" column, and print each price together with the **date** on which it occurred. If the same price occurs on several dates, print the first one you encounter while scanning the array from index 1 upward.

Expected output format

Your numbers will come from the file; the layout must match this sketch.

```
Weekly records: 574
From 1/7/08 to 12/31/18

Average price per year - Buffalo
2008: $3.50
2009: $?.??
...
2018: $?.??

Average price per month - New York City
Jan: $3.03
Feb: $?.??
...
Dec: $?.??

Highest NY State Average: $?.?? on M/D/YY
Lowest NY State Average:  $?.?? on M/D/YY
```

Hints.

- Split each row with `String[] fields = gasData[i].split(",");` (or a `StringTokenizer` with delimiter `","`). Then `fields[4]` is Buffalo and `fields[6]` is New York City.
- For the per-year and per-month averages, keep two small arrays: a running *sum* and a *count* for each year (or month). One pass over the data is enough.
- Test your date parsing on `1/7/08` *and* `12/31/18` before you trust it.

Before you submit – checklist

- ☐ All three files compile with no errors, and each has my name in its header comment.
- ☐ I did not change any of the existing template code (including `getData`).
- ☐ Q1: the four counts add up to the line length; both sample runs match exactly.
- ☐ Q2: all four examples match; result built with a `StringBuilder`; both letter cases handled.
- ☐ Q3: header row skipped; self-check values match; prices printed with two decimals.
- ☐ Zip named `LastName_FirstName_HW1.zip` uploaded to the D2L dropbox before the deadline.