

Project 1: TextFilter

Release date: **February 2nd, 2018**

Due date: **February 16th, 2018 by 11:59pm**

This is an individual project, to be completed on your own. It is considered dishonest either to read someone else's solution or to provide a classmate with a copy of your work. Do not make the mistake of thinking that superficial changes in a program (such as altering comments, changing variable names, or interchanging statements) can be used to avoid detection. If you cannot do the work yourself, it is extremely unlikely that you can succeed in disguising someone else's work. We are adamant that cheating in any form is not tolerated. Even the most trivial assignment is better not done than if you cheat to complete it.

Prerequisites

1. Primitives
 2. Strings
 3. Scanners
 4. Conditionals
 5. Loops
-

Learning Objectives

1. Critical Thinking
 2. Problem Solving
 3. Parsing and modifying complex Strings
-

Problem

Text manipulation is one of the most commonly needed routines in computer science. Consider a company like Twitter. They see millions of tweets every day, of all shapes and sizes. It may be incumbent upon them to censor out certain (vulgar) words in certain circumstances, or to provide text editing features to make composing tweets easier (such as find and replace). Also consider a game like World of Warcraft, which may want to censor out personal information (such as email addresses) from its in-game chat window in real-time, in order to prevent breaches of privacy.

Several years after graduating from Purdue, you and a group of your old buddies from college decide to make a startup social networking company. Looking back at what you learned, you think fondly of your days in CS180, and decide to implement the new platform using Java :)

After launching your network, your users have been outspoken at their delight for a new and unique experience. However, users would like to share information privately between other users - or to the world - and have tools available to filter that information as they see fit. Your company has decided to implement some of the most requested features as an addition to your social network, which are explained in more detail below.

This project is separated into 4 parts that you have to complete:

1. Part 1: Censoring Words
 2. Part 2: Replacing Words
 3. Part 3: Censoring Personal Information
 4. Part 4: Filtering multiple times
-

Part 1 - Censoring Words

In this part, you will be taking a **single, non-empty line** from the user, as well as a keyword (or key-phrase) from the user. You will then be replacing any instance of the keyword (or key-phrase) in the sentence with X's. Consider the following examples:

Example 1:

An Uncensored Passage:

I scream, you scream, **we all scream for ice cream!**

The same passage, but Censored with the keyword "we all scream for ice cream" :

I scream, you scream, **XXXXXXXXXXXXXXXXXXXXXXXXXXXXX!**

In this case, we are censoring the key-phrase, spaces and all, with X's

Example 2:

An Uncensored Passage:

How much **wood** would a woodchuck chuck if a woodchuck could chuck **wood**? He would chuck, he would, as much as he could, and chuck as much **wood** as a woodchuck would if a woodchuck could chuck **wood**.

The same passage, but Censored with the keyword "wood" :

How much **XXXX** would a woodchuck chuck if a woodchuck could chuck **XXXX**? He would chuck, he would, as much as he could, and chuck as much **XXXX** as a woodchuck would if a woodchuck could chuck **XXXX**.

In this phrase, we are searching for the word "wood" within the and replacing each instance of that word with X's.

Note:

- Words may be censored if they are surrounded by whitespace
- Words may be at the beginning and end of the String
- Words may be followed by a period, comma, question mark or exclamation point
 - (. , ? !)
 - This is only true if these characters come **after** the Word!
- Words that are part of a larger word should **not** be censored.
 - (i.e. “wood” and “woodchuck” from the example above)

Part 2 - Replacing Words

In a similar fashion to Part 1, in Part 2 you will be finding keywords or key-phrases in a sentence or passage. The only difference is that this time, you will also ask what the user wants to put in place of this keyword/key-phrase.

Consider the following example, which simply replaces “chickens” with “turtles” :

An Uncensored Passage:

Don’t count your **chickens** before the eggs have hatched!

The same passage, but Censored with keyword “chickens” and filter “turtles”

Don’t count your **turtles** before the eggs have hatched!

And a more complex example:

An Uncensored Passage:

I recently upgraded my desktop computer with a new **i7 processor**. Now I can render my **recording projects even faster!**

The same passage, but Censored.

I recently upgraded my desktop computer with a new **GTX 1080 graphics card**. Now I can play **my favorite games in 4K Ultra HD!**

Can you spot the difference?

Key-phrase → “**i7 processor**. Now I can render my **recording projects even faster!**”

Filter → “**GTX 1080 graphics card**. Now I can play my **favorite games in 4K Ultra HD!**”

Note: The same rules apply for spacing, location and special characters as in **Part 1!**

Part 3 - Censoring Personal Information

In the world today, online privacy is a serious concern. Your users have outcried at the leaks of personal information that hackers have posted for the world to see. Your mission in this task is to find that personal information and censor it before it even gets posted to your social network. Your users will be happy, leakers will be foiled, and privacy advocates will rejoice for your efforts!

This task requires careful attention to detail and formatting. Users will be entering lines that may contain personal information related to Names, Emails and Telephone Numbers. Your job is to parse through their input and censor that information accordingly.

In the skeleton code we have given you, we included the way that input strings should be formatted.

- Users will enter a line of information or text, and hit enter to add another line to the input.
 - After each time the User hits the enter key, we will add the new text to a string that will store all of the input.
 - After doing this, we will also add a newline character to separate the lines of input.
- Once the user enters an empty line, we will stop reading input.

For Example, if the user were to type in:

Name: Purdue Pete

Email: pete@purdue.edu

The String we would store, **including** newline characters, would be:

Name: Purdue Pete\nEmail: pete@purdue.edu\n

After obtaining input, it is your job to censor any personal information. First we will look at an example, and then outline what you explicitly need to censor.

An Uncensored Input:

Name: Purdue Pete

Email: pete@purdue.edu

Phone: 456-832-7180 // ILO - VEC - S180 :)

The newly Censored input:

Name: P***** **e

Email: p***@p*****.edu

Phone: ***-***-7180

Remember that each line has a newline character (\n) at the end of it!

When censoring information, we will be using the following format:

- "Type: Information_here\n"
 - Type we want to filter can be "Name", "Email" or "Phone"
 - There will always be a colon and a space between the "Type" and "Information_here"
 - Each line will end with a newline character (\n)
- When Censoring **Names**:
 - Only the first letter of the first name and last letter of the last name should remain uncensored
 - All other characters should be censored
 - The space between the first and last name should **NOT** be censored!
- When Censoring **Email Addresses**:
 - In the format <identifier>@<domain>.com:
 - Only the first letter of the **identifier** should remain uncensored
 - Only the first letter of the **domain** should remain uncensored
 - The extension, i.e. .com, .edu, .net, etc should remain uncensored
- When Censoring **Telephone Numbers**:
 - Only the last 4 digits should remain uncensored
- **All characters that are censored should be replaced with an asterisk (*)**

NOTE:

- **It is safe to assume that during testing, inputs will be always be given in this format and that they will not be malformed.**

Part 4 - Filtering multiple times

- For this part, you will simply need to update the value of the `boolean keepGoing` toward the bottom of the skeleton code.
 - **Hint: Make sure that the user enters something, i.e. wait for a non-empty line!**
-

Coding

This project is split up into four parts: Censoring and Replacing Words, Censoring Personal Information and Filtering multiple times. Each one of these parts was explained in detail above, and will require you to think about the reading, parsing and changing of Strings using the programming tools you have learned thus far.

If you have not read or skipped the instructions for Parts 1, 2, 3 and 4 above, please go back and read them before continuing!

You are being given a skeleton code that outlines what parts you need to implement and suggestions on where you should implement them.

- Please read over this file as well as the rest of the handout before you begin coding.
- Areas where you will need to implement your code have been marked with TODO comments.
 - **DO NOT alter the given skeleton code outside of these TODOs. Doing so may result in the failure of test cases during grading!**
- You can access the skeleton code in Vocareum.

Notes

- We suggest that you work in chronological order, that is, starting with Part 1 and ending with Part 4.
- If you are curious, or would like to use the powerful Java API String functions, you can view the String API here:
 - <https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

Grading Rubric

Part 1 - Censoring Words -- 30%

Part 2 - Replacing Words -- 30%

Part 3 - Censoring Personal Information -- 30%

Part 4 - Filtering multiple times -- 10%

Submission Instructions

- Log into Blackboard
- Select Project 01
- Upload your code

Your final submission must be in a compilable state. Code that does not compile will receive 0 points.

Good Luck

