CS/240/Lab/1

In the previous lab, you have implemented a simple character counter filter for a Twitter client. In this one, we will create two more filters – an email address anonymizer and a message purifier. You are not allowed to use any library functions except the I/O functions. This means you can only include `stdio.h` in your source files. More specifically, the only I/O functions you may use are: `getchar`, `putchar`, `scanf`, and `printf`.

Q1: Anonymizer

Anonymizer is used to identify all email addresses in a message and blank the user name part out for preventing leaking private informations. A real world email addresses usually look like `<user_name>@<domain_name>`. However, handling the varieties of user name requires sophisticated algorithms, so we are going to be simplistic in anonymizer:

- user names must be alphanumeric words followed immediately by a `@`.
- user names must be recognized in their maximum possible length as long as legal.
- user names must be anonymized by replacing every character in it with an `X`.
- domain names must not be anonymized.

> To be precise, we define the user name again in regular expressions:
>
> `<user_name> = [a-zA-Z0-9]+`
>
> You are not required to fully understand regular expressions, but the definition here is simple. Informally, it says a user name is a non-empty string consisting of only letters or digits. A little more formally, it says that the string is generated by starting from an empty string and appending the characters chosen from the square brackets (lowercase or capital letters or digits here) repeatedly and at least once (which is what the plus sign means). For example, words like `000`, `abc`, `1a2b3c` are all legal user names, while `1a.2b` or `a$b` are not due to the forbidden characters `.` and `$`.

In terms of legal user names with the maximum length, here is an example: for the message `Hi, my email address is h.potter@hogwarts.edu.`, the user name should be recognized as `potter` rather than any other strings like `h.potter`, `otter`, etc. If you scan the user name right to left starting from `@`, `otter` is not the longest legal while `h.potter` contains illegal character dot.

To clarify, if the input message is:

```
Hi, my email addresses are h.potter@hogwarts.edu and 1a@2b3c@.
```

Then the output should be:

```
Hi, my email addresses are h.XXXXXX@hogwarts.edu and XX@XXXX@.
```

If the input message is:

```
Although it's not an email address, I'd hate if@you saw my secret@ word.
```

Then the output should be:

```
Although it's not an email address, I'd hate XX@you saw my XXXXXX@ word.
```

Hints

The code to test if the variable `c` is alphanumeric is :

```
if( c >= 'A' && c <='Z' || c >= 'a' && c <='z' || c >= '0' && c <='9') { ... }
```

Compiling

```
gcc -std=c99 -o anonymizer anonymizer.c
```

Testing

```
cat msg.txt | ./anonymizer   OR   ./anonymizer < msg.txt
```

Q2: Purifier

Purifier is introduced to kill bad words in messages. However, there are more swears and foul words than minutes in the day. We can only censor against a few of them, which are listed below:

- `darn`
- `fool`
- `hell`

Note that, the keywords must be whole word matches, that is, no letters should precede or follow a keyword immediately. Put it another way, any ASCII characters other than letters are considered potential separators separating the keywords from the rest of the message. For example, the `hell` in `go to hell`! is considered a keyword because it is separated by the space on the left and ! on the right, while in `hello world`! it is not.

For simplicity, the match is case-sensitive – you only have to catch lower case versions of each word. The purifier should locate all the keywords and replace each character of them with an `X`. If the input is:

```
hello, may the fool "darn" go to hell!
hello, may the fool (darn) go to hell again!
hell, may the fool $darn$ go to hell again and again!
```

the output should be:

```
hello, may the XXXX "XXXX" go to XXXX!
hello, may the XXXX (XXXX) go to XXXX again!
XXXX, may the XXXX $XXXX$ go to XXXX again and again!
```

Hints

Without using library functions, the only way to test the equivalence between two strings is to 1) make sure them have the same length; 2) compare them character by character. You can choose to store the keywords in a `char*` or char[] array (see the lecture notes) and iterate over each of them when comparing a target string against the list (the text book has examples).

Compiling

```
gcc -std=c99 -o purifier purifier.c
```

Testing

```
cat msg.txt | ./purifier   OR   ./purifier < msg.txt
```

Turning in

You will be assigned a unique turnin ID sent in an email to your @purdue.edu email address. Go to the course web page (`http://www.cs.purdue.edu/homes/jv/courses/240s12`), and under the "submissions" section click "web site". On the submission web page, enter your unique turnin ID in the box and press "log in". Under "currently open projects" will be listed "lab 1 anonymizer" and "lab 1 purifier". Click "lab 1 anonymizer", use the file selector to choose your `anonymizer.c` file, then click "submit". The page will tell you how you've done, and your total score. When you're satisfied, click "return to list of projects", then "lab 1 purifier". Use the file selector to choose your `purifier.c` file, then click "submit". When you're satisfied, close your browser window. You may resubmit at any time before the due date by the same process.

Lab is due Monday, January 16th before midnight. No late labs accepted.

Grading criteria

Anonymizer

- source file is named anonymizer.c
- code compiles
- code runs without error
- code only includes stdio.h
- able to blank user names and keep the rest of the message in original form
- able to blank all email addresses with all varieties of user names without false positive
- able to handle multi-line messages

Purifier

- source file is named purifier.c
- code compiles
- code runs without error
- code only includes stdio.h
- able to blank bad words and keep the rest of the message in original form
- able to blank all keywords without false positive or false negatives
- able to handle multi-line messages