SEng 265 — Introduction to Software Engineering
Fall 2002
Assignment No. 2

Note 1 **This assignment is to be done in teams of two.**

Note 2 Your assignment will be tested on aserver.

- Due date: Oct 9, at the beginning of the class.
- This assignment is worth 20 points.
- This assignment is worth 5% of your total course mark.
- Clearly mark student numbers on all submissions.

Objectives

After completing this assignment, you will have:

- learn the basics of Perl and regular expressions.

Introduction

E-mail has become one of the two killer applications of the Internet (the other being the World-Wide Web). It is surprising how little support some e-mail clients provide to perform complex searches in mail boxes.

The RFC 822 (http://www.faqs.org/rfcs/rfc822.html) defines the format that mail messages should have. In a nutshell, this format is very simple (figure 1 shows a mail message that demonstrates this format):

- A mail message is defined as a **header** followed by a **body**. The separator between these two is two consecutive empty lines.
- A **header** is composed of a sequence of **fields**.
- Each **field** is composed of a **field name**, followed by a `:` (colon) followed by its value. If the field is too long it can span several lines, as long as these lines start with one or more spaces or tabs.
- Field names are not case sensitive, and some names are standardized: **From**, **To**, **Date**, etc.
- Finally, in the `mbox` format (still prevalent in many email clients such as pine, elm, mail, vm, etc) each mail message starts with a line of the form:

  ```
  From <sender> <date>
  ```

  and ends with two consecutive line-feeds (the line-feeds are considered to belong to the body of the message). `<sender>` is the address of the sender, and `date` is the date when the message was sent. E-mail servers and clients guarantee that no line can start with the string "`From `" (to test this assertion, email yourself a message that includes "From dmg" –remember, the line should start with the F).

As mail keeps piling up, a tool to scan my email messages is urgently needed.

```
From dmg@al.cs.uvic.ca  Mon Sep 23 16:19:34 2002
Message-Id: <200209232319.g8NNJTK10926@al.cs.uvic.ca>
Errors-To: "Daniel M. German" <dmgerman@uvic.ca>
X-UVic-Virus-Scanned: OK - Passed virus scan by Sophos v3.59 (sophie) on cascara
X-Scanned-By: MIMEDefang 2.19 (www . roaringpenguin . com / mimedefang)
From: "Daniel M. German"
    <dmgerman@uvic.ca>
To: dmg@turingmachine.org
Subject: Testing
Date: Mon, 23 Sep 2002 16:19:29 -0700


This is a message to show how simple the mail format is. The following
2 lines are part of the message I am sending:

From: dmgerman@uvic.ca
To: dmgerman@uvic.ca

--
Daniel M. German                  "Programming today is a race
                                   between software engineers
                                   striving to build bigger
                                   and better idiot-proof
                                   programs and the universe
                                   trying to produce bigger and
                                   better idiots.
    Richard Cook  ->               So far, the universe is winning. "
http://turingmachine.org/
dmgerman@uvic.ca
```

Figure 1: An example of a mail message

Your task, should you choose to accept it:

I want you to write the command: `postal_digger`. This is its man page:

```
postal_digger(1)

NAME
       postal_digger - search email messages in the old, ancient MBOX
       format.

SYNOPSIS
       postal_digger [OPTIONS] [filename]

DESCRIPTION

        postal_digger uses its OPTIONS to determine what to search for
        in a mail file (see description of options below). If filename
        is specified, it reads its input from that file, otherwise
        postal_digger reads its
        input from stdin.

        postal_digger provides options to scan some of the different
        fields in the header of the message and the body. It uses the
        power of Perl regular expressions to find matches.

        In the descriptions of options below a PATTERN is a valid perl
        regular expression, such as /abc/, abc, a+b+c+, ^defg$, [a-z],
        etc.

        postal_digger, by default, should treat any sequence of
        whitespace (spaces, tabs, or line feeds) like a single
        space. For instance, when searching for "the party" should
        return any message in which the words "the" and "party" appear
        in that order regardless of how many whitespaces there are
        between both words (for this example, there must exist at
```

least one). This is done for both, the headers and the body of the message.

Without any option, postal digger writes every single message in its input to the standard output (i.e. it behaves like cat). postal_digger can restrict the information printed to one or more of the following fields: subject, from, to, date, and the body of the message.

When the user specifies one or more of the searching pattern's options (--to, --body, --subject, --from) a 'match' is defined as a message that matches all different options at the same time.

OPTIONS

--casesensitive
By default postal_digger is case insensitive. This option forces the search to be case sensitive.

--preservewhitespace
Do not treat sequences of whitespace as a single space, i.e. treat each whitespace as in the original message ("the party" will not match "the party")

--from=PATTERN
search value of the "from" field for the PATTERN. --from='@.*uvic.ca' will scan for messages that come from any UVic account.

--subject=PATTERN
search the subject field for the corresponding PATTERN.

--to=PATTERN
search value of the "to" for the PATTERN. --to='@.*uvic.ca' will scan for messages that come from any UVic account.

--body=PATTERN
search the body of the message for PATTERN.

--printsubject
--printfrom
--printto
--printdate
--printbody
By default postal_digger prints the entire message that is matched. These options force it to print the "Subject" "From", "To" or "Date" fields, or the body of the message. The field should appear exactly as it appears in the original message. If more than one of these options is specified, the order in which the fields is printed should be the order in which their corresponding

options appears above.

When printing the body of the message, the empty line that separates the body from the header should be included.

--invert
Invert the sense of matching, to select non-matching messages.

--count
Print to standard error a line of the following form, showing the number of messages matched and the number of messages scanned:

End of search. 3 message(s) matching out of 20.

NOTES:

postal_digger is able to handle inputs of any size.

In a given mail message, some fields are optional. When postal_digger is searching for a value in a given field, and the field does not exist, that message is not considered a match.

The search should be case insensitive unless otherwise specified.

In every option, PATTERN is a valid perl regular expression.

postal_digger should print a short usage message to stderr when an invalid option is specified or when it is run without any parameters.

EXAMPLES:

Print the subject of those messages that include the address dmgerman@uvic.ca in the from field and talked about seng265.

```
postal_digger --from='\bdmgerman@uvic.ca\b' --body='seng ?265'
```

Print the date of those messages that talk about 'Jim Smith'

```
postal_digger --printdate --body='Jim Smith\b' --casesensitive
```

Print the entire message that includes in the body something that looks like a telephone number (7 digits, potentially separated after the 3rd digit by a space or a hyphen):

```
    postal_digger  --body='[0-9]{3}[ \-]?[0-9]{4}'
```

Print only those messages in which the string dmgerman@uvic.ca is included in the to field:

```
    postal_digger --to='dmgerman@uvic.ca' ''
```

Print those messages that include a subdomain of uvic.ca in the from field:

```
    postal_digger --from='@[^.@]+\.uvic.ca'
```

Print the subjects of all messages:

```
    postal_digger --print_subject
```

Print only those messages that do not include the word software in its body:

```
    postal_digger --body='software' --invert
```

Count the number of messages in the input:

```
    postal_digger --count
```

KNOWN BUGS

* postal_digger does not handle MIME encodings. It treats them as a sequence of characters.

* postal_digger does not interpret any header field (MIME encodings in headers, for example). It is only able to remove whitespace from it.

* postal_digger assumes that the input conforms to the expected format.

* postal_digger is only able to accept each different option at most once

AUTHOR

Written by you.

REPORTING BUGS

Report bugs to <you@uvic.ca>.

COPYRIGHT

Implementation Notes

- The same option cannot appear more than once in the same command line. For instance, you cannot run it as:

  ```
  postal_digger --printsubject --printsubject testfile
  ```

- There are no restrictions on how to implement this command but you should use a reasonable amount of memory and CPU.
- Your program can safely assume that all the input conforms to the expected format.
- You **must** follow our Perl coding standards (where applicable). Any disparity or question will be resolved in the bulletin board.
- No line in your source code should exceed 80 characters long!
- You can find an example of an mbox mail file at `/home/seng265/assign2/testing_mbox`
- We will use only this file for testing.
- Only Perl modules installed in the SENG machines (e.g. aserver) will be allowed!
- Your code should be commented, at least according to our coding standards (discussed in class and in the bulletin board)
- Design your solution in a way that you can program it incrementally. A solution that handles some options is better than a solution that fails trying to handle all solutions. Your mark will depend on how many features your program supports correctly. At the very least your program should handle the case in which no parameters are specified!
- My solution contains slightly less than 300 LOCs (not counting comments).

Do you think this is too easy?

For those who get bored by this assignment, I offer some extra marks for the team that *flawlessly* (without bugs) is able to either:

- handle searching in HTML mail messages (ignores tag names) –easy 10% extra;
- handle decoding of MIME messages and searching in only the text parts –difficult, can use other Perl modules, 20% extra.

30% for the team that can handle both! If you are interested in this bounty, then check the bulletin board for further information.

What to submit

- Submit your source code (`postal_digger`) via CVS in a directory called `assign2` directory of the module of each of the members of the team (yes, that means that both members should do the check-in the files).
- At the start of the class on the deadline, **hand in a printed copy of your `cvs commit` script as well as a hard copy of your source code file**.

Remember, no line in your source code should exceed 80 characters long!