Assignment 2: speed, Speed

version: 0.3 last updated: 2021-03-12 18:30

It is possible in response to student queries that there will be minor changes to this specification, most likely in subset 2,
This may entail minor changes to work you have already completed..

Autotests are incomplete. More will be added.

Aims

This assignment aims to give you

- practice in Perl programming generally
- a clear concrete understanding of sed's core semantics

Note: the material in the lecture notes will not be sufficient by itself to allow you to complete this assignment. You may need to search on-line documentation for Perl, Sed, etc. Being able to search documentation efficiently for the information you need is a *very* useful skill for any kind of computing work.

Introduction

Your task in this assignment is to implement **Speed**, a subset of the important Unix/Linux tool Sed.

You will do this in Perl hence the name **Speed**

Sed is a *very* complex program which has many commands. You will implement only a few of the most important commands. You will also be given a number of simplifying assumptions, which make your task easier.

Speed is a subset of POSIX-compatible `sed` with extended regular expressions. On a CSE systems you would run **`sed --posix -r`**

You must implement **Speed** in Perl only. See the **Permitted Languages** section below for more information.

Reference implementation

Many aspects of this assignment are not fully specified in this document; instead, you must match the behaviour of the reference implementation: **`2041 speed`**

Provision of a reference implementation is a common method to provide or define an operational specification, and it's something you will likely need to do after you leave UNSW.

Discovering and matching the reference implementation's behaviour is deliberately part of the assignment, and will take some thought.

If you discover what you believe to be a bug in the reference implementation, report it in the class forum. Andrew may fix the bug, or indicate that you do not need to match the reference implementation's behaviour in this case.

Speed Commands

Subset 0

In subset 0 `speed.pl` will always be given a single Speed command as a command-line argument.

The sed command will be one of 'p','q' and 's' (see below).

The only other command-line argument possible in subset 0 is the **`-n`** option.

Input files will not be specified in subset 0. For subset 0 `speed.pl` need only read lines from standard input.

Subset 0: addresses

Speed commands can optionally be preceded by an address specifying the line(s) they apply to,

In subset 0, this address can either be line number or a regex.

The line number must be a positive integer.

The regex must be delimited with slash `/` characters.

Subset 0 & 1: Regexes

In subset 0 and 1, you can assume backslash `\` does **not** appear in address regexes

In subset 0 and 1, you can assume backslash `\` does **not** appear in address regexes.

In subset 0 and 1, you can assume the character used to delimit the regex can **not** appear in the regex itself.

In subset 0, regexes are delimited with slash `/` characters so you can assume slashes do not appear in regexes.

In all subsets, you can assume the regex is correct. You do not have to check for errors in the regex.

In all subsets, you can assume the regex is compatible with Perl. In other words, the regex can be used as Perl regular expressions and will have the same meaning..

Subset 0: q - quit command

The Speed **`q`** command causes `speed.pl` to exit, for example:

```
$ seq 1 5|2041 speed 3q
1
2
3
$ seq 10 15|2041 speed /.1/q
10
11
```

Subset 0: p - print command

The Speed **`p`** commands prints the input line, for example:

```
$ seq 1 5|2041 speed 2p
1
2
2
3
4
5
```

Subset 0: d - delete command

The Speed **`d`** commands deletes the input line, for example:

```
$ seq 1 5|2041 speed 4d
1
2
3
5
```

Subset 0: s - substitute command

The Speed **`s`** command replaces the specified regex on the input line.

In subset 0, the regex must be delimited with slash `/` characters.

In subset 1, you can assume that slash `/` characters do not appear in the regex or replacement string.

In subset 0 & 1, you can assume the replacement string is compatible with Perl and can be used in a Perl substitute command with the same effect.

```
$ seq 1 5|2041 speed 's/[15]/zzz/'
zzz
2
3
4
zzz
```

The substitute command can followed optionally by the modifier character **`g`**, for example:

```
$ echo Hello Andrew|2041 speed 's/e//'
Hllo Andrew
$ echo Hello Andrew|2041 speed 's/e//g'
Hllo Andrw
```

`g` is the only permitted modifier character.

Subset 0: -n command line option

The Speed **`-n`** command line option stops input lines being printed by default.

```
$ seq 1 5|2041 speed -n 3p
3
```

Subset 1

Subset 1 is more difficult. You will need to spend some time understanding the semantics (meaning) of these operations, by running the reference implementation, or researching the equivalent **`sed`** operations.

Note the assessment scheme recognises this difficulty.

Subset 1: addresses

In subset 1, **`$`** can be used as an address. It matches the last line, for example

```
$ seq 1 5|2041 speed '$d'
1
2
3
4
```

In subset 1, Speed commands can optionally be preceded by a comma separated pair of address specifying the start and finish of the range of lines the command applies to, for example:

```
$ seq 10 21|2041 speed 3,/2/d
10
11
21
```

In subset 0 and 1, slash **`/`** and backslash **`\`** may appear in address regexes.

Subset 1: s - substitute command

In subset 1, any non-whitespace character may be used to delimit a substitute command, for example

```
$ seq 1 5|2041 speed 'sX[15]XzzzX'
zzz
2
3
4
zzz
```

In subset 0 & 1, you can assume the replacement string is compatible with Perl and can be used in a Perl substitute command with the same effect.

In subset 1, you can assume that the character used to delimit the substitute command does not appear in the regex or replacement string.

Subset 1: Multiple Commands

In subset 1, multiple Speed commands can be supplied separated by semicolons **`;`**, for example:

```
$ seq 1 5|2041 speed '4q;/2/d'
1
3
4
```

In subset 1, you can assume semicolons do not occur anywhere inside Speed commands

Subset 1: -f command line option

The Speed **`-f`** reads Speed commands from the specified file, for example

```
$ echo 4q >commands.speed
$ echo /2/d >>commands.speed
$ seq 1 5|2041 speed -f commands.speed
1
3
4
```

Subset 1: Input Files

In subset 1, input files can be specified on the command line:

```
$ seq 1 2 >two.txt
$ seq 1 5 >five.txt
$ 2041 speed '4q;/2/d' two.txt five.txt
1
1
2
```

Subset 1: Comments & White Space

In subset 1, whitespace can appear before and after commands & addresses and '#' can be used as a comment character, for example:

```
$ seq 24 42|2041 speed ' 3, 17  d  # comment'
24
25
41
42
```

In subset 1, you can assume newline does not appear in Speed commands specified as a command line argument.

Subset 2

Subset 2 is even more difficult. You will need to spend considerable time understanding the semantics of these operations, by running the reference implementation, and/or researching the equivalent **sed** operations.

Note the assessment scheme recognises this difficulty.

Subset 2: Regexes

In subset 2, backslash **** may appear in regexes.

In subset 2, the character used to delimit the regex may appear in the regex itself.

Subset 2: s - substitute command

In subset 2 you can not assume, that the replacement string is compatible with Perl and you can not assume it can be used in a Perl substitute command with the same effect.

In subset 2, the character used to delimit the substitute command may appear in the regex or replacement string.

In subset 2, backslash may appear in the regex or replacement string.

Subset 2: -i command line option

The Speed **-i** command line options replaces file contents with the output of the Speed commands. You should use a temporary file.

```
$ seq 1 5 >five.txt
$ cat five.txt
1
2
3
4
5
$ 2041 speed -i /[24]/d five.txt
$ cat five.txt
1
3
5
```

Subset 2: Multiple Commands

In subset 2, semicolons can appear inside Speed commands.

```
$ echo 'Punctuation characters include . , ; :'|2041 speed 's/;/semicolon/g;/;/q'
Punctuation characters include . , semicolon :
```

In subset 2, newline can be used to separate Speed commands passed a command-line argument.

Subset 2: : - label command

The Speed **:** command indicates where **b** and **t** commands should continue execution.

There can not be an address before a label command.

Subset 2: b - branch command

The Speed **b** command branches to the specified label, if the label is omitted, it branches to end of the script.

Subset 2: t - conditional branch command

The Speed **t** command behaves the same as the **b** command except it branches only if there has been a successful substitute command since the last input line was read and since the last **t** command.

Subset 2: a - append command

The Speed **a** command appends the specified text.

Subset 2: i - insert command

The Speed **i** command inserts the specified text.

There can not be an address before an insert command.

Subset 2: c - change command

The Speed **c** command replaces the selected lines with the specified text.

Other Sed Features

You do not have to implement in Speed sed features and commands other than those described above.

For example, sed on CSE systems provides extra commands including **[]DhHgGlnpPtvWxy** which are not part of Speed

For example, sed on CSE systems provides extra commands including `{} D n H g G l n p r w W x y` which are not part of Speed.

For example, sed on CSE systems adds extra syntax to addresses including feature involving the characters: `! + ~ 0 \`. These are not part of Speed.

For example, sed on CSE systems has a number of command-line options other than `-i`, `-n` and `-f`. These are not part of Speed

The reference implementation implements many of these extra sed features and commands.

The marking will not test your code on these these extra features and commands.

You do not have to check for these extra features and commands.

You will not be penalized if you choose to implement any of these extra features and commands.

Diary

You must keep notes on each piece of work you make on this assignment. The notes should include date, starting and finishing time, and a brief description of the work carried out. For example:

Date	Start	Stop	Activity	Comments
19/06/19	16:00	17:30	coding	implemented basic -n functionality
20/06/19	20:00	10:30	debugging	found bug in command-line arguments

Include these notes in the files you submit as an ASCII file named `diary.txt`.

Testing

Autotests

As usual, some autotests will be available:

```
$ 2041 autotest speed speed.pl
...
```

You can also run only tests for a particular subset or an individual test:

```
$ 2041 autotest speed subset1 speed.pl
...
$ 2041 autotest speed subset1_13 speed.pl
...
```

If you are using extra Perl (.pl or .pm) files, include them on the autotest command line.

You can download the files used by autotest as a zip file or a tar file.

You will need to do most of the testing yourself.

Test Scripts

You should submit ten Shell scripts, named `test00.sh` to `test09.sh`, which run speed commands that test an aspect of Speed.

Your test script should check whether the test is passed or failed and print a suitable message.

Your test script should exit with status 0 if the test was passed and exit with status 0 if it was failed.

The `test??.sh` scripts do not have to be examples that your program implements successfully.

You may share your test examples with your friends, but the ones you submit must be your own creation.

The test scripts should show how you've thought about testing carefully.

You are only expected to write test scripts testing parts of Speed you have attempted to implement. For example, if you have not attempted subset 2 you are not expected to write test scripts testing .

Permitted Languages

Your programs must be written entirely in Perl.

Start `speed.pl` with:

```
#!/usr/bin/perl -w
```

If you want to run your code on a machine `perl` is not installed in `/usr/bin/` you can alternatively start `speed.pl` like this:

```
#!/usr/bin/env perl
use warnings;
```

NOTE:

> Your answer must be Perl only. You can not use other languages such as Shell, Python or C.
>
> You may **not** run external programs, e.g. via system or backquotes. For example, you can't run `cat`, `head`, `tail`.
>
> You are permitted to use any Perl module installed on CSE servers.
>
> Your Perl code should work with `/usr/bin/perl` on CSE servers.
>
> Your Perl code should not generate warnings when run with `/usr/bin/perl -w` on CSE servers.
>
> You may submit extra `.pl` or `.pm` Perl files.

Assumptions/Clarifications

Like all good programmers, you should make as few assumptions as possible.

You can assume that only the arguments described above are supplied to `speed` commands. You do not have to handle other arguments.

You must apply the Speed commands to input lines as you read the input lines. You can not read all input lines into array There may be an unlimited number of input lines.

You should match the output streams used by the reference implementations. It writes error messages to stderr: so should you.

You should match the exit status used by the reference implementation. It exits with status 1 after an error: so should you.

You can assume the directory containing your Perl files is in the environment variable `$PERL5LIB`.

You can assume arguments will be in the position and order shown in the usage message from the reference implementation. Other orders and positions will not be tested. Here is the usage message:

```
$ ./speed.pl --help
usage: speed.py [-i] [-n] [-f <script-file> | <sed-command>] [<files>...]
```

You can assume, Speed regular expressions and valid Perl regular expressions and are compatible with Perl. In other words, they can be used as Perl regular expressions and will have the same effect.

You can assume command line arguments, STDIN and all files contain only ASCII bytes.

You can assume all input lines in `STDIN` and in all files are terminated by a `'\n'` byte.

Speed error messages include the program name. It is recommended you use **`$0`** however it also acceptable to hard-code the program name. The automarking and style marking will accept both.

Change Log

Version 0.1
(2021-03-10 12:00)
- Initial release

Version 0.2
(2021-03-12 14:00)
- $ as address added
- prohibition of eval removed
- clarification of subsets added

Version 0.3
(2021-03-12 18:30)
- change assignment name to Speed to avoid connection to derogratory term

Assessment

Testing

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest speed
```

`2041 autotest` will not test everything.
Always do your own testing.

Automarking will be run by the lecturer after the submission deadline, using a superset of tests to those `autotest` runs for you.

Submission

When you are finished working on the assignment, you must submit your work by running `give`:

```
$ give cs2041 ass2_speed speed.pl diary.txt test??.sh [any-other-files]
```

You must run `give` before **Monday Apr 26 09:00 2021** to obtain the marks for this assignment. Note that this is an individual exercise, the work you submit with `give` must be entirely your own

exercise, the work you submit with `give` must be entirely your own.

You can run `give` multiple times.
Only your last submission will be marked.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

You *cannot* obtain marks by e-mailing your code to tutors or lecturers.

You can check your latest submission on CSE servers with:

```
$ COMP(2041|9044) classrun -check ass2_speed
```

You can check the files you have submitted here.

Manual marking will be done by your tutor, who will mark for style and readability, as described in the **Assessment** section below. After your tutor has assessed your work, you can view your results here; The resulting mark will also be available via give's web interface.

Due Date

This assignment is due **Monday Apr 26 09:00 2021**.

If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 2%. For example, if an assignment worth 74% was submitted 10 hours late, the late submission would have no effect. If the same assignment was submitted 15 hours late, it would be awarded 70%, the maximum mark it can achieve at that time.

Assessment Scheme

This assignment will contribute 15 marks to your final COMP(2041|9044) mark

15% of the marks for assignment 2 will come from hand-marking. These marks will be awarded on the basis of clarity, commenting, elegance and style: in other words, you will be assessed on how easy it is for a human to read and understand your program.

5% of the marks for assignment 2 will be based on the test suite you submit.

80% of the marks for assignment 2 will come from the performance of your code on a large series of tests.

An indicative assessment scheme follows. The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

HD (85+)	All subsets working; code is beautiful; great test suite and diary
DN (75+)	Subset 1 working; good clear code; good test suite and diary
CR (65+)	Subset 0 working; good clear code; good test suite and diary
PS (55+)	Subset 0 passing some tests; code is reasonably readable; reasonable test suite and diary
PS (50+)	Good progress on assignment, but not passing autotests
0%	knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 FL for COMP(2041|9044)	submitting any other person's work; this includes joint work.
academic misconduct	submitting another person's work without their consent; paying another person to do work for you.

Intermediate Versions of Work

You are required to submit intermediate versions of your assignment.

Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the `give` command below. It is fine if intermediate versions do not compile or otherwise fail submission tests. Only the final submitted version of your assignment will be marked.

All these intermediate versions of your work will be placed in a Git repository and made available to you via a web interface at `https://gitlab.cse.unsw.edu.au/`*`z5555555`*`/21T1-comp2041-ass2_speed` (replacing *z5555555* with your own zID). This will allow you to retrieve earlier versions of your code if needed.

Attribution of Work

This is an individual assignment.

The work you submit must be entirely your own work, apart from any exceptions explicitly included in the assignment specification above. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted.

You are only permitted to request help with the assignment in the course forum, help sessions, or from the teaching staff (the lecturer(s) and tutors) of COMP(2041|9044).

Do not provide or show your assignment work to any other person (including by posting it on the forum), apart from the teaching staff of COMP(2041|9044). If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted, you may be penalized, even if that work was submitted without your knowledge or consent; this may apply even if your work is submitted by a third party unknown to you. You will not be penalized if your work is taken without your consent or knowledge.

Do not place your assignment work in online repositories such as github or any where else that is publically accessible. You may use a private repository.

Submissions that violate these conditions will be penalised. Penalties may include negative marks, automatic failure of the course, and possibly other academic discipline. We are also required to report acts of plagiarism or other student misconduct: if students involved hold scholarships, this may result in a loss of the scholarship. This may also result in the loss of a student visa.

Assignment submissions will be examined, both automatically and manually, for such submissions.

Change Log

COMP(2041|9044) 21T1: Software Construction is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au
CRICOS Provider 00098G