DAT 103
Datamaskiner og operativsystemer (Computers and Operating Systems)

Obligatory Assignment – Words Reversal.

The assignment will demonstrate:

- How a simple algorithm is run at the hardware level, i.e. how high-level program code translates to assembly. To this end, you will implement a function that reverses the order of words in a string.
- How the Unix shell can be used to automate tasks and to tie low-level programs (like this string reversal) together to do more complicated work. In the example, this entails pre- and post-processing of the input, and storage of auxiliary data.

This project consists of two parts:

Part I: A bash script that reads input, splits it up, uses standard tools to categorize it and writes/reads some files, in order to make it simpler to reverse the words.

Part II: An assembly program that does the simplified reversal operation.

The behaviour desired from the complete program, i.e., Part I and Part II together, is best understood from an example: the text

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum
```

should in the end be transformed into

```
laborum est id anim mollit deserunt
officia qui culpa in sunt, proident non cupidatat occaecat
sint Excepteur. pariatur nulla fugiat eu dolore cillum esse
velit voluptate in reprehenderit in dolor irure aute Duis.

consequat commodo ea ex aliquip ut nisi laboris ullamco
exercitation nostrud quis, veniam minim ad enim Ut. aliqua
magna dolore et labore ut incididunt tempor eiusmod do sed,
elit adipiscing consectetur, amet sit dolor ipsum Lorem
```

Notice how the punctuation marks and newlines are between the same words as they were in the original version, which is different behaviour from what a naïve split on space characters would produce.

Part II will implement the naïve space-split in assembly. Part I is responsible for preprocessing the punctuation.

Part I: Bash Wrapper

Deadline: 23.9.2022, 23:59

Your task for Assignment Part I is to write a Bash script `words-reverse.sh` that takes text from STDIN. It will then isolate any punctuation substrings, in order to obtain a simplified form of the text which can then be processed by the low-level string reverser.

Subtask 1 Command Experiments

When writing shell scripts, it is usually most effective to first experiment on the command line how to use existing tools to do parts of the work. For this section, you should demonstrate this by carrying out some primitive file manipulations and text transformations, and documenting what you have done with screenshots from your terminal window (cf. Figure 2). Make a screenshot for every occurence of ⌈📷 n⌉ in the following, and save it as `screenshots/`n`.png`.

Figure 1: Example of roughly how your screenshots may look. Ensure that (1) it is visible what the command in question was, that (2) both the command and either the whole resulting output (if any) or ca. 5 lines from it are fully visible, and that (3) not too much unrelated content is in frame.

1A Directories and Files

For the purpose of experimenting with given text data, set up a scratch directory whose name includes your student number ⌈📷 0⌉. In that directory, create (without using an editor) a text file containing the first paragraph of the Lorem Ipsum example (as given above) ⌈📷 1⌉. Check that the file was indeed created ⌈📷 2⌉, and that it contains the intended text ⌈📷 3⌉. Note that you should accomplish all these tasks with bash commands.

1B Newlines / Separator Characters

Many shell tools work most easily on input that is chunked by lines, however our data contains newline characters which we would like to treat as just another punctuation mark. Therefore, as a first step, use a Posix tool to replace those newlines with the marker `|` (which you can assume will never occur in the input to your program). I.e., you should run a command whose output is of the form

```
Lorem ipsum...elit,|sed do eiusmod...magna|aliqua...
```

⌈📷 4⌉

(If you like, you can store this intermediate result in another file and use that in the following, or alternatively use always the original file and already build up a processing pipeline. This is a matter of taste.)

Next, split the text again up in lines, each containing alternately a single word or a word-separating space/punctuation string. By "word" we understand here a sequence of letter characters (lower- or uppercase characters from the Latin/English alphabet).

⌈📷 5⌉ (See an example screenshot in Figure 2)

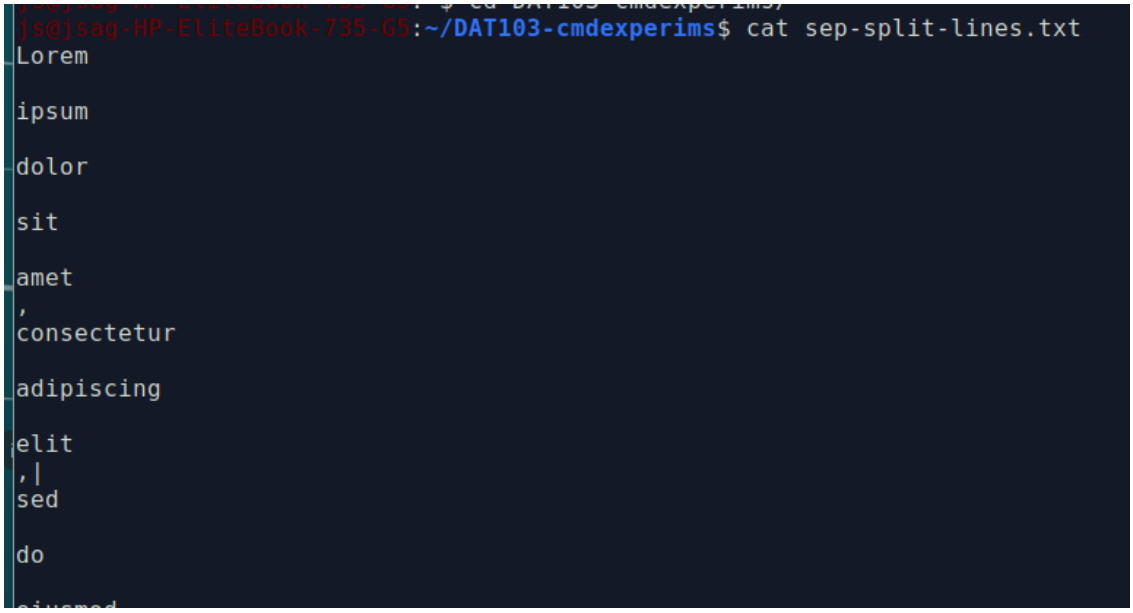

Figure 2: How the text should look after splitting up the words and separators into individual lines.

1C Hash substitution

Notice that the punctuated-string-reversal could now be accomplished by inverting the order of *lines*, and then applying the inverse of the transformations from this section. However, we impose an additional restriction: that the low-level reverser cannot directly deal with punctuation, and that punctuation therefore needs to be substituted with numbers first.[1]

Specifically, you are to substitute each punctuation string by its cryptographic hash (256-bit SHA-2).

```
...
0a07f659461970d8d8dcefe4fff96a1745599810dc7d47391e95c803b7b2072c
consectetur
36a9e7f1c95b82ffb99743e0c5c4ce95d83c9a430aac59f84ef3cbfab6145068
adipiscing
36a9e7f1c95b82ffb99743e0c5c4ce95d83c9a430aac59f84ef3cbfab6145068
elit
5f7b4d4963cf7747ec54d37065ccdcfea4ec23ab58ba20e5db891f52602a2b05
sed
...
```

Hashing is technically speaking not an invertible substitution, but in practice the hashes are easily unique, so you just need a lookup map. In this case, the file system is used as the lookup: for each punctuation string a file with the hash as its name is created.

Test this out on the command line: store one of the punctuations in a shell variable⌈📷 6⌉, and then devise a single command that will generate a file containing that string, with the hash as the file name⌈📷 7⌉. Test this for several different punctuations, then check that the corresponding files have been created⌈📷 8⌉ and have the intended content⌈📷 9⌉.

[1]The restriction is admittedly somewhat artificial, since the punctuation characters are also just ASCII and could easily be processed directly by an assembly program. A more realistic reason for this kind of transformation would be if the input could contain arbitrary Unicode (in UTF-8), because that is a variable-length encoding and thus more difficult to handle in assembly.

Subtask 2 Depunctuation Script

Now automate the commands you explored in the previous section. Write a script `depunctuate.sh` that

- Takes one command-line argument: the directory in which to store the hash lookups.
- Reads a text from standard input.
- Generates all required lookup files in the directory.
- Outputs the preprocessed text, in the form of lines alternating between single words and punctuation-hashes as.

It should be a standard executable Bash script. Make a reasonable effort to structure it to be as clear as possible, and comment anything nontrivial, explaining what is happening and why you did it this way.

Before generating the lookup for each punctuation, the script should always check if such a file is already there. If yes, it should not overwrite it but instead check whether the file already contains this punctuation. If yes, nothing needs to be done here, but if the file already contains something different it means either a user error or a hash collision has occured. You should handle this case by terminating **with a clear error**.

Subtask 3 Repunctuation Script

In a similar vein, write a script `repunctuate.sh` that inverses the transformation of `depunctuate.sh`, i.e. it

- Takes the same command-line argument: now the directory in which to *search* for hash-lookup files.
- Reads a text from standard input.
- Uses the lookup files to substitute back the original punctuation.
- Re-assembles the original line breaks.

Again, this should be a standard, executable, commented Bash script.

Subtask 4 Putting it All Together

Now write the script `words-reverse.sh`, which

- Takes no command-line arguments, or the single argument `--bypass`. Any other arguments should result in an error.
- Creates a *unique temporary directory* for the hash lookups.
- Passes standard input to `depunctuate.sh`, with the created directory as the argument.
- Passes its output to the low-level reverser `words-reverse-ll` (since that will only be implemented in the next assignment, you can not test this part yet), unless invoked with `--bypass`, in which case this step is skipped or passed to `cat` instead.
- Passes this output to `repunctuate.sh`, again with the lookup directory.
- Gives back this final result on standard output.

- Cleans up the hash lookups.

This script, when run with the `--bypass` option, should give back its original input (because the substitutions are inverses of each other). Test this extensively, with both the provided example text and input of your own choice[📷 10].

Obligatory Submission Part I (due on 23.9.2022, 23:59)

When/Where to submit:

- You will have make your submission in Canvas on/before 23.9.2022, 23:59.

How to submit:

- You can work in a group of at most 3 people. Note that if you work in a group, the group should be formed and registered in Canvas **before the submission**. If you join a group after the submission, a new submission has to be made in order to receive the grade of the assignment.
- Pack the source code you wrote in **a single archive file** called `oblig1-`*studnr*`.tar`, where *studnr* is your student number. If you work in a group, use the group ID instead: `oblig1-group`*grpID*`.tar`.

  For example, `oblig1-4567.tar` if student 4567 submits alone, or `oblig1-group89` for group 89.

  (Please avoid including any spaces, uppercase or non-ASCII characters in the file name. The separator should be *a single hyphen/minus character*, as is good practice in any Unix project.)

  This archive must contain exactly the following:

  1. `words-reverse.sh`,
  2. `depunctuate.sh`,
  3. `repunctuate.sh`, and
  4. a directory `screenshots` with numbered `.png` files in it.

  Double-check that your archive conforms to the requirements, by running the sanity checker program that we will provide in week 37. We may refuse to grade or perhaps even look at assignments that do not succeed the sanity check.