Assignment 6: Category-Partition

In this individual assignment, you must generate **between 60 and 100 test-case specifications** (i.e., generated test frames) for a simplified version of the `replace` utility, whose specs are provided below, using the category-partition method that we saw in class. Make sure, when defining your test specifications, to suitably cover the domain of the application under test. Finally, **make sure to suitably use constraints (rather than eliminating choices) to keep the number of test cases within the specified thresholds**.

Concise Specification of the `replace` Utility

- NAME:
  `replace` - looks for all occurrences of string *from* and replaces it with string *to*. It is possible to specify one or more files on which to perform the replace operation(s) in a single replace command.
- SYNOPSIS
  `replace OPT <from> <to> -- <filename> [<filename>]*`

  where `OPT` can be **zero or more** of
  - `-b`
  - `-f`
  - `-l`
  - `-i`
- COMMAND-LINE ARGUMENTS AND OPTIONS

  `from`: string to be replaced with string `to`.

  `to`: string that will replace string `from`.

  `filename`: the file(s) on which the replace operation has to be performed.

  `-b`: if specified, the `replace` utility creates a backup copy of each file on which a replace operation is performed before modifying it.

  `-f`: if specified, the `replace` utility only replaces the first occurrence of string `from` in each file.

  `-l`: if specified, the `replace` utility only replaces the last occurrence of string `from` in each file.

  `-i`: if specified, the `replace` utility performs the search for string `from` in a case insensitive way.

- EXAMPLES OF USAGE

  Example 1:
  ```
  replace -i Howdy Hello -- file1.txt file2.txt file3.txt
  ```

  would replace occurrences of "Howdy" with "Hello" in the files specified. Because the "`-i`" option is specified, occurrences of "howdy", "HOwdy", "HOWDY", and so on would also be replaced.

  Example 2:
  ```
  replace -b -f Bill William -- file1.txt file2.txt
  ```

  would replace the first occurrence of "Bill" with "William" in the two files specified. It would also create a backup copy of the two files before performing the replace.

  Example 3:
  ```
  replace -f -l abc ABC -- file1.txt
  ```

  would replace both the first and the last occurrences of "abc" with "ABC" in the file specified.

Notes

- As stated above, in defining your categories and choices, you should make sure to suitably cover the domain of the application under test. This also includes possibly erroneous inputs. Just to give you an example, if you had to test a calculator, you may want to cover the case of a division by zero.
- You are only required to specify test inputs for the application, but **you do not have to also specify the expected outcome for such inputs**. It is therefore OK if you don't know how the system would behave for a specific input. Using the same calculator example, you could test the case of a division by zero even if you do not know how exactly the calculator would behave for that input.

Tools and Useful Files

You will use the `TSLgenerator` tool to generate test frames starting from a TSL file, just like we did in the demo. A version of the `TSLgenerator` tool for Linux, Mac OS X, and Windows (two versions), together with a user manual, are available at:

- TSLgenerator-manual.txt
- TSLgenerator.linux
- TSLgenerator-mac
- TSLgenerator-win32.exe
- TSLgenerator-win64.exe

Important: **These are command-line tools**, which means that you have to run them from the command line, as I did in my demo, rather than by clicking on them. Also, on Linux and

Mac systems, you may need to change the permissions of the files to make them executable using the `chmod` utility. To run the tool on a Mac, for instance, you should do the following, from a terminal:

```
chmod +x TSLgenerator-mac
./TSLgenerator-mac <command line arguments>
```

Finally, if you are running a modern version of Windows (> XP), you may want to try the `TSLgenerator-win64.exe` version of the tool. It hasn't been tested thoroughly, but it should work better on these versions than the `TSLgenerator-win32.exe` version. If you encounter issues, please post a public question on Piazza and consider running the tool on a different platform (if you have the option to do so).

We are also providing file split-example.txt, the TSL file for the example we saw in the lesson, for your reference.

Also, the demo example for your benefit.

How To Submit

- Create a directory "`Assignment6`" in your personal GitHub repo.
- Add to this new directory two text files:
  - `catpart.txt`: the TSL file you created.
  - `catpart.txt.tsl`: the test specifications generated by the `TSLgenerator` tool when run on your TSL file.
- Commit and push your files to GitHub.
- Submit on Blackboard the commit ID for your submission.