Lab 6: Filemaker

Introduction

Lab 6 is our final Python programming project and a fun one called `filemaker`. Your `filemaker` script is to create a file containing a specified number of records based on a record layout. The record layout consists of constants and commands that when read and executed by your Python script, will cause a file to be created that matches that format.

Requirements

Your Python script shall be named `filemaker`, located in the root of your repo and be marked executable.

```
Usage: ./filemaker INPUTCOMMANDFILE OUTPUTFILE RECORDCOUNT
```

The INPUTCOMMANDFILE contains one command per line in the file. Your filemaker script shall support the following commands in the INPUTCOMMANDFILE:

`HEADER "string"`	writes "string" once at the top of the file
`STRING "string"`	writes the constant "string" to each record in the output
`WORD label "filename"`	write a random word from file "filename"
`INTEGER label min max`	write a random integer between min and max inclusive
`REFER label`	write the item referred to by label (see notes below)

The filenames and labels in commandfiles are randomly assigned. Do not hard-code them in your script.

The label identifier you see above is a string that is unique in this file from all other labels and is a way to reference something created earlier. For example, consider the following sample command file we will call `commandfile1`:

COMMAND	EXPLANATION
`STRING '"'`	Output a double quote (notice that it is a single quote followed by a double quote followed by a single quote)
`WORD lastname "surnames.txt"`	Output a random word from file "surnames.txt", labeled lastname
`STRING ", "`	Output a comma followed by a space
`WORD firstname "firstnames.txt"`	Output a random word from file "firstnames.txt", labeled firstname
`STRING '","'`	Output a double quote, comma and a double qoute
`REFER firstname`	Output the same random word in this record as the command labeled firstname
`STRING "."`	Output a period
`REFER lastname`	Output the same random word in this record as the command labeled lastname
`STRING '@mail.weber.edu"\n'`	Output `@mail.weber.edu` followed by a double quote and a newline

If all input files were available and you were to execute this command:

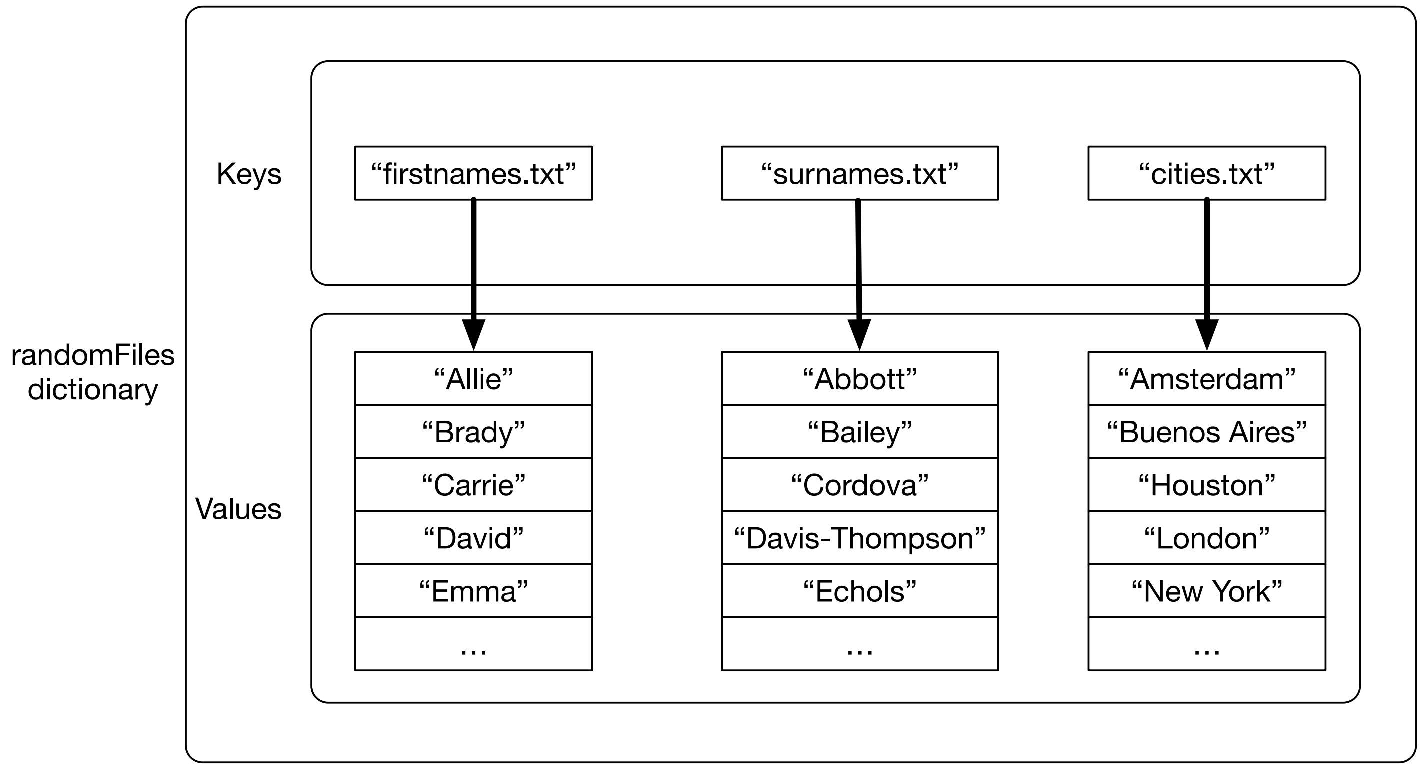
```
./filemaker commandfile1 outputfile 3
```

Your script should produce a file called `outputfile` with three lines in it, with content similar to this:

```
"Cowan, Ted","Ted.Cowan@mail.weber.edu"
"Mouse, Mickey","Mickey.Mouse@mail.weber.edu"
"Duck, Donald","Donald.Duck@mail.weber.edu"
```

A Word About Dictionaries

In the Helpful Hints section below, reference is made to a dictionary you might create that holds data read in from files referenced in the commandfile, with file names like "firstnames.txt" and "surnames.txt". In the commandfile, the file name can be any valid file name, so **don't assume these specific names.** A picture might help to explain this concept.

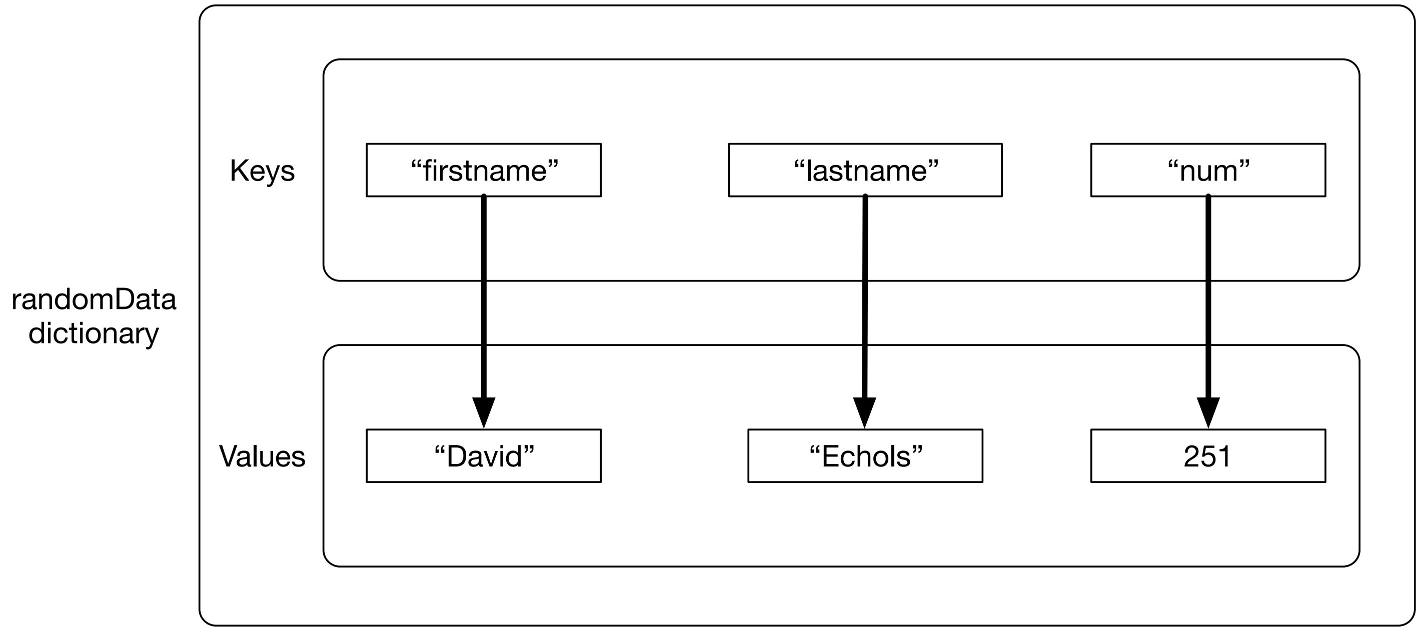


An example dictionary `randomFiles` *storing the contents of text files*

In this example, dictionary `randomFiles` has been populated with three keys, "firstnames.txt", "surnames.txt" and "cities.txt". In each case, the name of a file is the **key** and the entire contents of that file is an array stored as the **value** of that key.

To store the entire file in the dictionary as an array, use the `readlines()` method. Then, to obtain a random entry from one of the arrays, generate a random number to use as the index to that array.

A second reference is made to a dictionary called `randomData`, which is a holding place for data randomly retrieved from `randomFiles` or generated using the random number generator. The label of the WORD or INTEGER command is used as the key and the associated random data is stored as the value in the `randomData` dictionary. Here is an example of what `randomData` might contain:



Helpful Hints

- Do **not** use the Python function `split()` to parse the commandfile. Use `shlex.split()` as described in the module video.
- Use `try/except` when opening any files or when converting RECORDCOUNT to an integer.
- The suggested logic below works, but you may see places where you can optimize. **Resist the temptation** to optimize your code until you have it working perfectly.
- The module video might suggest there is extra credit in this lab for completing certain features. This is an error and I apologize. All features are required for full credit.
- The `.decode("string_escape")` string function works only with Python2. For Python3, use `.encode("utf-8").decode("unicode_escape")`.

Suggested logic:

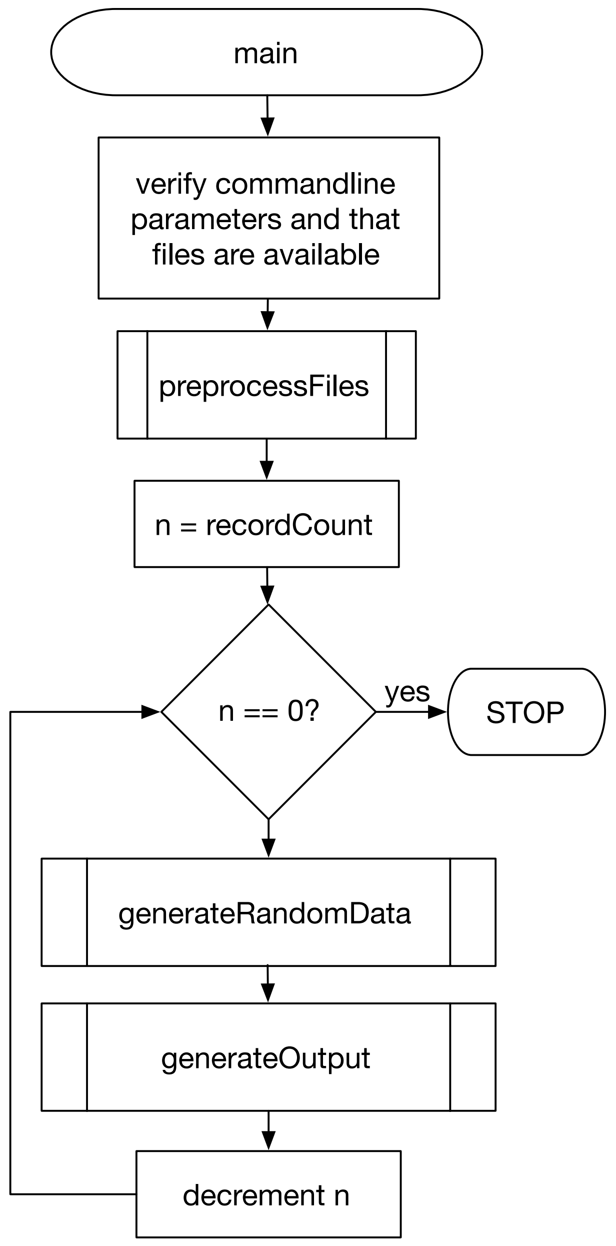
- **Verify commandline parameters and files:** Do this only once.
  - Issue an `Usage` message and `exit(1)` if the user did not specify all three parameters.
  - Convert RECORDCOUNT from a string to an integer (call it `recordCount`) and `exit(1)` if not numeric or not a positive integer.
  - Issue an appropriate `Error` message and `exit(1)` if an error occurs opening any of the files.
- **Preprocess files:** Do this only once. For each line in the command file:
  - `HEADER`: write the header string to the output file
  - `WORD`: "slurp" the entire file as a single array into a dictionary called `randomFiles`, using `readlines()`. Use the file name as the key. `exit (1)` if the file is not readable.
- **Loop RECORDCOUNT times to generate output**:

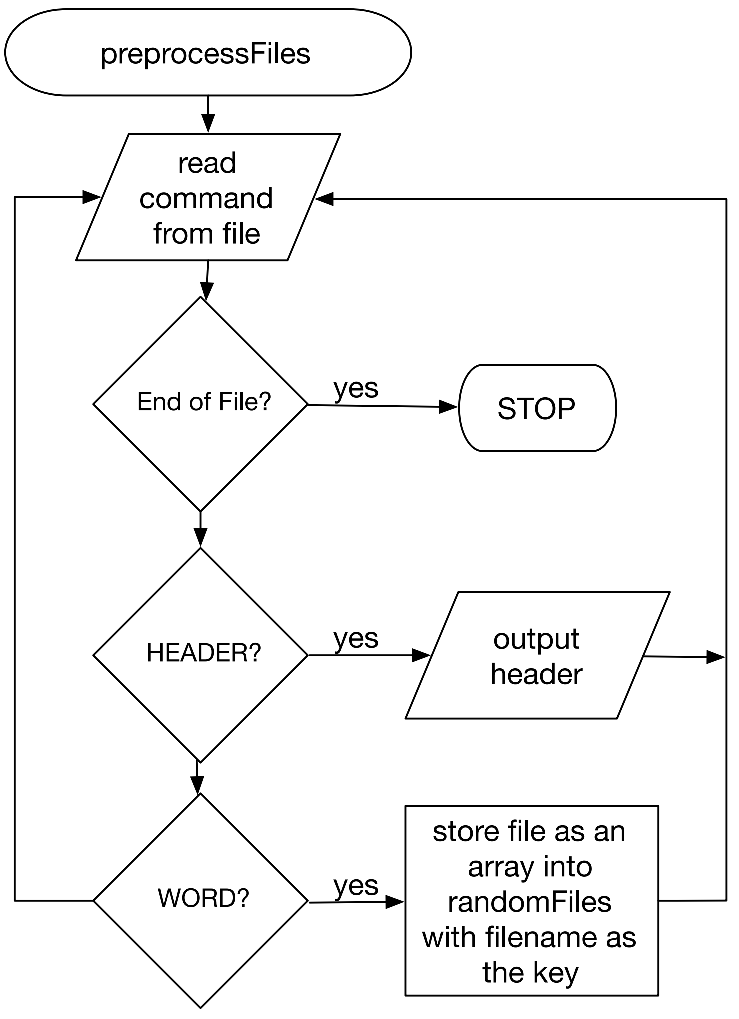
- Initialize dictionary `randomData`
    - **Generate random data** which will populate dictionary `randomData` with random words and integers. See detail below.
    - **Generate output** which will output strings, as well as random words from files and random integers from dictionary `randomData`, which were generated in the previous step. See detail below.

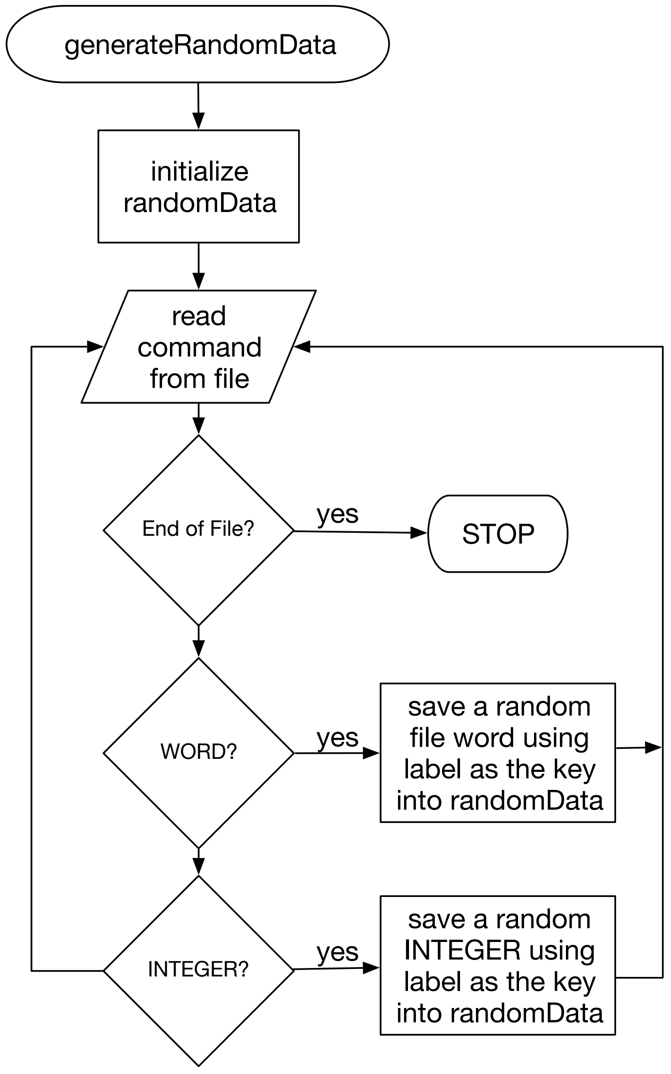
- **Generate random data:** For each command in the the command file:
    - `WORD`: use a random number as a subscript to the array of words. The name of the file is the key to the array of words in dictionary `randomFiles`. Use `0` as the `min` and `len(randomFiles[label])-1` as the `max`. Store the word in `randomData` with the label as the key.
    - `INTEGER`: generate a random integer using random.randint(min,max). Store the integer in `randomData` with the label as the key.

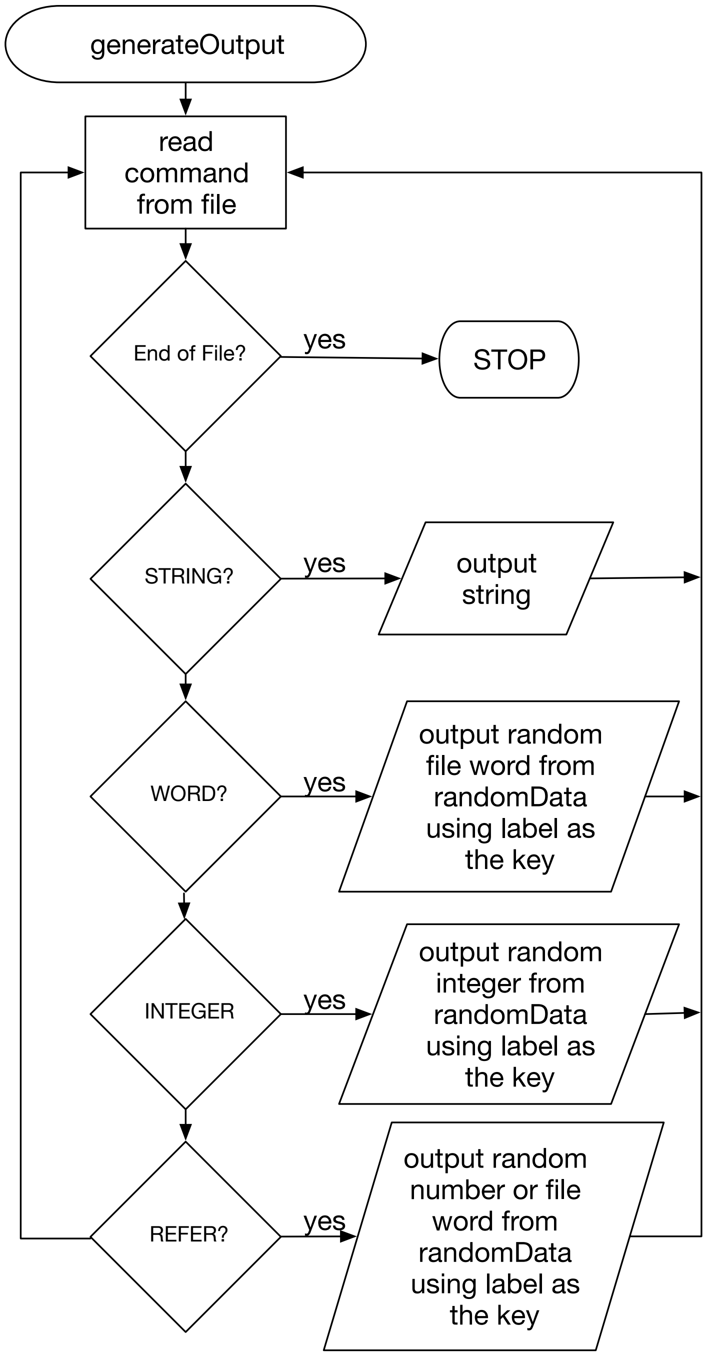
- **Generate output** - for each command in the command file:
    - `STRING`: write the string to the output file
    - `INTEGER`: write to the output file the random integer stored in `randomData` using the label as the key
    - `WORD`: write to the file the random word stored in `randomData` using the label as the key
    - `REFER`: write to the file the random data stored in `randomData` using the label as the key

These diagrams below may make the above logic a bit clearer.


main
verify commandline parameters and that files are available
preprocessFiles
n = recordCount
n == 0?
yes
STOP
generateRandomData
generateOutput
decrement n


preprocessFiles
read command from file
End of File?
yes
STOP
HEADER?
yes
output header
WORD?
yes
store file as an array into randomFiles with filename as the key


generateRandomData
initialize randomData
read command from file
End of File?
yes
STOP
WORD?
yes
save a random file word using label as the key into randomData
INTEGER?
yes
save a random INTEGER using label as the key into randomData


generateOutput
read command from file
End of File?
yes
STOP
STRING?
yes
output string
WORD?
yes
output random file word from randomData using label as the key
INTEGER
yes
output random integer from randomData using label as the key
REFER?
yes
output random number or file word from randomData using label as the key

Clone your private repo on github.com

In the assignment module in Canvas, find the link to the Canvas page entitled "Clone your Lab 6 Repo", click on it and follow the instructions.

Copy your private repo down to your Linux system

```bash
git clone https://github.com/cowancs3030<SEMESTER>/lab6-YOURGITHUBUSERNAME

cd lab6-YOURGITHUBUSERNAME
```

Write and test `filemaker`

Fire up your favorite text editor, and update the header:

```
#!/usr/bin/python3
# (Your name)
# Lab 6 - Filemaker
# CS 3030 - Scripting Languages

(add your mind-blowingly kewl code here)
```

Test files provided for your convenience

When manually testing your script, the following files are available in your repo for you to use in testing:

`firstnames.txt`	a file of first names, one per line
`surnames.txt`	a file of surnames (last names), one per line
`lab6cmds`	the command file used in the example above
`lab6cmds2`	the command file above plus HEADER and INTEGER

Test your `filemaker` script with these files like this:

```
./filemaker lab6cmds out.txt 3
```

```
./filemaker lab6cmds2 out2.txt 3
```

Your script should create `out.txt` and `out2.txt` with the appropriate content. `lab6cmds2` contains references to `firstnames.txt` and `surnames.txt`. Study these files carefully.

After you have tested your code manually and confirmed it is working correctly, run cucumber to check your progress:

```
cucumber
```

⚠ cucumber randomly generates testfiles so you will want to run cucumber many, many times to verify your script's operation.

Submit your assignment code for grading

⚠ Remember, you must push your script in your private repo to github.com to receive any points, for all assignments in this course.

```bash
git add filemaker
git commit -m"COMMIT MESSAGE"
git push origin master
```

For this lab you will have created the following executable files:

```
filemaker
```

Grading

Here is how you earn points for this assignment:

FEATURES	POINTS
Must-Have Features	
Script is named correctly and found in its proper place in your private repo	5
Script is executable	5
Required Features	
Script prints a “Usage:” statement and exits rc=1 if any of the commandline parameters are missing	10
Script prints an error message containing the word “Error” and exits rc=1 if the INPUTCOMMANDFILE file cannot be opened	10
Script prints an error message containing the word “Error” and exits rc=1 if the OUTPUTFILE file cannot be opened	10
Script prints an error message containing the word “Error” and exits rc=1 if the COUNT is negative or not a number	10
Script exits rc=0 on successful completion	10
Script outputs the requested number of records	30
Script supports HEADER	30
Script supports STRING	40
Script supports WORD	20
Script supports WORD with true randomness	20
Script supports INTEGER	20
Script supports INTEGER with true randomness	20
Script supports REFER to a label on a WORD command	30
Script supports REFER to a label on a INTEGER command	30
Grand Total	300