Name: Catherine Healy, chealy5@nd.edu

Link to Github Repository:
https://github.com/CatherineH3/Theory_Extra_Credit/tree/main

What was the extra work:

I wrote a program that takes a CFG and puts it in a simplified format called Chomsky Normal Form. I selected this project to improve my understanding of CFGs. See "*Additional Details*" section of this document for more details on the implementation of this program.

I also created input / test files, and I generated output finals for the final form to check my algorithm.

Why did I choose it:
I felt deficient in my understanding of CFGs and the rules of production. Problems like Homework 7A problem one were very confusing to me. I did not understand how to construct grammar or how to make the parse tree.
This work helped improve my understanding of production rules, terminals and nonterminals, recursive production rules, and how epsilon fits into the production rules.
The idea for this project was suggested by Professor Kogge.

File descriptions

test_chealycsv: This contains a CSV test file in the format specified by the "file format" section below. I got this test case by looking at the geeksforgeeks tutorial here:
https://www.geeksforgeeks.org/converting-context-free-grammar-chomsky-normal-form/
It also helped me make my algorithm

output_chealy.txt: This is the output from my Chomsky normal form code. It is formatted nicely so the production rules make sense.

chomsky_normal_form_catherine_healy.py: This is the code to convert a grammar into Chomsky normal form. I used the algorithm described in the additional details section. It also is very clearly commented.

trace_chomsky_catherine_healy.py: This is a program written to trace the changes of made to the CFG. It uses the same algorithm as chomsky_normal_form_catherine_healy.py and outputs trace_output_chealy.txt

trace_output_chealy5.txt: This file is the output that shows the trace of the changes made to the CFG. All of the steps to convert the CFG to Chomsky normal form are outlined

Additional Details:

From Textbook Page 108, Chomsky normal for is a simplified for for context free grammars. It is useful in giving algorithms for context-free grammars.
A context-free grammar is in Chomsky normal form if every rule is in the form:
A → BC
A → a
Where *a* is any terminal and A, B, and C are any variables– except B and C may not be the start variable.
Additionally, the rule S → ε is permitted, where s is the start variable

Algorithm Background:
Any CFG can be converted into Chomsky Normal Form.
The following steps are involved:

- Eliminate all epsilon rules, such as A → epsilon
- Eliminate all unit rules in the form A→ B
- Convert the Remaining Rules into the final form.

My Algorithm

- Add new start nonterminal production rule S0 --> S
- Remove all epsilon productions X --> e. Must account for this by adding a rule for everything else with that nonterminal with the nonterminal missing.
- Remove all unit productions (these are any single capital letters on the right side). Replace by finding all instances of the single nonterminal, removing it, and appending it with the entire right side to the places where there is unit production (exception: if recursive, don't do this_
- Look for terminals mixed with nonterminals
- If there is not a rule for a terminal going to a signle nonterminal (ex: X--> a), add them. Then, go and replace the nonterminal with that rule so S-->AaS is S-->AXS
- look for two terminals in a row. Replace them with some nonterminals. (Can use repeats) A--> bb goes to A-->VV is V --> b
- Look for more than 2 Nonterminals. A-->XAS Take the first two and make a new rule that points to the first one. Z --> XA Then replace the first two with that rule and you can keep the first the same A--> ZS

Input File format:

Note about input file format: I used "e" to represent all epsilons to avoid issues with the program being able to detect these symbols. I assumed that there was one symbol on the right, so the input file had the form. Other than the first character, which represents the character on the right, Strings separated by a comma are representative of strings separated by |. There is one rule per line. Below is an example:

To represent the following CFG form textbook page 110:
S → ASA | B
A → B | S
B → b

The following input csv is given:
S, ASA, aB
A, B, S
B, b, e

T simply the coding, I will assume that the start always starts with an S and that all left-hand side parts of the production rule are unique characters (repeats are combined to an “or” logic with |), so that I can use the dictionary data structure to store the rules.

Verification of Output:
I based my test on an example from a geeks for geeks tutorial (https://www.geeksforgeeks.org/converting-context-free-grammar-chomsky-normal-form/)
Here is the tutorial output:

```
S0-> AS|PB| SB
S → AS|QB| SB
A → RS|XS|a
B → TS|VV|US|XS|a
X → a
Y → b
V → b
P → AS
Q → AS
R → XA
T → SY
U → XA
```

Here is my program output (note the order and variable names are different but it describes the same CFG.

```
S0 --> SB | AS | TB
S --> SB | AS | WB
A --> a | ZS | VS
B --> a | ZAS | ZS | XX | US
Z --> a
Y --> b
X --> b
W --> AS
V --> ZA
U --> SY
```

T --> AS

Resources Used:

To help understand the algorithm, and construct the input files for my test cases, I used the following:

Geeks for Geeks Tutorial:
https://www.geeksforgeeks.org/converting-context-free-grammar-chomsky-normal-form/

Sipser “Introduction to the Theory of Computation” page 110