Follow the **guidelines** of the previous homework for packaging, submission, and allowable use of resources. Use the newsgroup for questions, clarifications, and discussion in general.

The **goal** of this assignment is to deepen our understanding of context-free grammars by implementing some algorithms found in proofs we have studied. We will continue to enhance the *Lexaard* language from previous assignments for this purpose.

1. (50 pts.) Extend Lexaard to parse and print context-free grammars (CFGs). In particular, extend the `define` and `print` statements to work with CFGs. You must fully parse the CFG, not just store it in a form that allows printing.

   The Lexaard representation of CFGs is very similar to the conventional representation used in the textbook and elsewhere, and is composed of a token `cfg` followed by a newline, followed by a name and optional comments on a single line, in turn followed by lines of the following kinds, in possibly mixed order:

   - Rule lines of the form: *h -> b1 b2 ... bp*
   - Variables-listing lines of the form: *v1 v2 ... vq*
   - Terminals-listing lines of the form: *.. t1 t2 ... tr*

   For example, the following are representations of the grammar G_1 and a few rules of the grammar G_2 from Section 2.1 of the textbook.

```
cfg
G1 p102
A -> 0 A 1
A -> B
B -> #
```

```
cfg
G2excerpt from Ex 2.10 of the textbook
sentence -> noun-phrase verb-phrase
noun-phrase -> cmplx-noun | cmplx-noun prep-phrase
noun -> boy | girl | flower
```

   The optional variables-listing lines allow us to represent CFGs that have one or more variables that do not appear on the left-hand side of any rule. Similarly, the optional terminals-listing lines permits the inclusion of terminals that do not occur in any rule. For example, the CFG `odd1` below has no rules for the variables `E`, `F`, and `G`; `odd2` has nonterminals `x` and `y` that do not occur in rules. The CFG `odd3` includes both variables and terminals that do not occur in rules. More examples: The CFG `g` below is shortest by representation length, while `h` is shortest of those with a nonempty language. CFG `why0`, though weird, is valid.

```
cfg
odd1 xtra v
A -> B
B -> C
C -> A
D -> D
E B F G
```

```
cfg
odd3 xtra t
A -> B b C
B -> A a
C -> A | ..
.. x y a
```

```
cfg
odd3 xtra vt
A -> B b C
B -> A a
C -> A | ..
E B F
.. x y a
```

```
cfg
g
S
```

```
cfg
h
S -> a
```

```
cfg
why0
S -> S
```

To simplify tokenizing, all tokens in a CFG in Lexaard are separated by whitespace. Thus, we write `A -> A a | B b b` instead of `A -> Aa|Bbb`. The start variable is always the first variable encountered in the representation, which is either the left-hand side of the first rule or, in case there are no rules, the first variable listed on the first variables line. Tokens appearing on the left-hand side of some rule, or on a variables-listing line, are variables while the rest, excluding `->` and `|`, are terminals.

When CFGs are printed, rules should be listed in lexicographic order, except that all rules for the start state are listed first. Rules are ordered lexicographically by lifting the lexicographic order on their constituent tokens, treated as strings of characters ordered by their ASCII codes. Further, all rules for the same head must be printed in a compact one-line representation with bodies separated by `|`. The rule bodies are ordered lexicographically. A single variables-listing line should be printed immediately after the rules iff there are variables that do not appear on the left-hand side of any rules, and it should list only those variables, in sorted order. An exception to this rule is when the variables line contains the start symbol; in this case (only), the variables line must be printed before any rules. Similarly, a single terminals-listing line should be printed as the last line iff there are terminals that do not appear in rules, and it should list only those terminals, in sorted order after the `..` prefix token.

2. (100 pts.) Extend Lexaard with a function `chomskyNF` that converts the CFG given as its argument to an equivalent CFG in Chomsky Normal Form, using the algorithm outlined in the proof of Theorem 2.9 in the textbook. This function is accessed in Lexaard in a manner analogous to that for functions described in earlier assignments.

3. (50 pts.) Extend Lexaard with a boolean function `cfgGen` that evaluates to true iff the CFG named by its first argument generates the string named as its second argument, using the method outlined in the proof of Theorem 4.7 in the textbook.

4. $\star$ (25 pts.) Extend Lexaard with a boolean function `cfgGenF` that is logically equivalent to `cfgGen` of Question 3 but that uses the method outlined in the proof of Theorem 7.16 in the textbook. Run suitable experiments to quantify the relative performance of `cfgGen` and `cfgGenF`. Include a clear description of the experiments and results, in tabular and chart form, in a PDF file packaged with the rest of the submission.