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 and pushdown automata 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 pushdown automata (PDAs). In particular, extend the `define` and `print` statements to work with PDAs. You must fully parse the PDA, not just store it in a form that allows printing.

   The Lexaard representation of PDAs is similar to the representation of FSAs used in earlier assignments. It consists of the token `pda` followed by a newline, in turn followed by a name and optional comments on a single line. As for FSAs, the rest of the representation encodes the transition function and other information in a tabular format. The main difference is that PDA representations have two alphabet rows instead of one: The first row lists the input alphabet and the second lists the stack alphabet. Following the two alphabet rows is a list of rows, one for each state, describing the transition function. As for FSAs, the state in the first row is the start state and accepting states are prefixed with a `*`. Each transition row consists of the name of the state (with possible `*` prefix) followed by mn whitespace-separated tokens, where m is the size of the stack alphabet and n is the size of the input alphabet. Each token encodes a set of (state, stack-operation) pairs by simply listing all such pairs separated by commas. In such tokens, the sequence `..` denotes ϵ, except when the token consists only of a single `..`, in which case it denotes $\emptyset$.

   The following is a representation of the PDAs M_1 from Example 2.15 of the textbook. By design, it is very similar to the table on page 114 of the textbook, which describes the transition function in the same manner. For instance, the entry `q4,..` in the penultimate row indicates $\delta(q_3, \epsilon, \$) = (q_4, \epsilon)$.

```
pda
M1 PDA of Example 2.14 on p 114 of textbook
       0                1                ..
       0   $    ..
*q1    ..   ..   ..      ..      ..  ..    ..  ..      q2,$
 q2    ..   ..   q2,0    q3,..   ..  ..    ..  ..      ..
 q3    ..   ..   ..      q3,..   ..  ..    ..  q4,..   ..
*q4    ..   ..   ..      ..      ..  ..    ..  ..      ..
```

   Repetition of symbols is permitted in alphabet rows. This feature may be used to make the table more human-friendly, as illustrated by the following representation of the PDA M_2 from Example 2.16 of the textbook.

```
pda
M2 PDA of Example 2.16 on p 116 of textbook
       a   a   a       b       b   b         c   c       c         ..   ..      ..
       a   $   ..      a       $   ..        a   $       ..        a    $       ..
 q1    ..  ..  ..      ..      ..  ..        ..  ..      ..        ..   ..      q2,$
 q2    ..  ..  q2,a    ..      ..  ..        ..  ..      ..        ..   ..      q3,..,q5,..
 q3    ..  ..  ..      q3,..   ..  ..        ..      ..  ..        ..   q4,..   ..
*q4    ..  ..  ..      ..      ..  ..        ..      ..  q4,..     ..   ..      ..
 q5    ..  ..  ..      ..      ..  q5,..     ..      ..  ..        ..   ..      q6,..
 q6    ..  ..  ..      ..      ..  ..        q6,..   ..  ..        ..   q7,..   ..
*q7    ..  ..  ..      ..      ..  ..        ..      ..  ..        ..   ..      ..
```

When PDAs are printed, each alphabet row should list symbols in lexicographic order. The transition rows should be in lexicographic order by state names, except that the row for the start state is always first. The table formed by the alphabet and transition rows should be aligned as illustrated by the example for M_2 above. However, these printing conventions are strictly optional for the input, so that the following is an alternate, valid, though less readable, representation of M_1.

```
pda
M1 PDA of Example 2.14 on p 114 of textbook
0 1 ..
0 $ ..
*q1 .. .. .. .. .. .. .. .. q2,$
q2 .. .. q2,0 q3,.. .. .. .. .. ..
q3 .. .. .. q3,.. .. .. .. q4,.. ..
*q4 .. .. .. .. .. .. .. .. ..
```

2. (80 pts.) Extend Lexaard with a function `cfg2pda` that converts the CFG given as its argument to an equivalent PDA using the textbook's method (Lemma 2.21, p. 117). This function is accessed in Lexaard in a manner analogous to that for functions described in earlier assignments. A simple test run could be as follows, where the material in square brackets is filled in suitably:

```
define g1 cfg
[... rest of G1 from HW04 ...]
define p1 cfg2pda g1
print p1
[... suitable PDA representation ...]
```

3. (70 pts.) Extend Lexaard with a function `pda2cfg` that converts the PDA given as its argument to an equivalent CFG using the textbook's method (Lemma 2.27, p. 121).

4. $\star$ (25 pts.) Augment the experimental study from the $\star$ question of HW04 to include a version of `cfgGen` that uses the above transformation of CFGs to PDAs. Present your results as before.