Exercise 1. (Exercise 9.6 - Speech and Language Processing) (3points)
Assume a grammar that has differently subcategorized verb rules:

- **Verb-with-NP-complement** → find | leave | repeat | ...
- **Verb-with-S-complement** → think | believe | say | ...
- **Verb-with-Inf-VP-complement** → want | try | need | ...

and many VPs rules for different subcategorization:

- **VP** → **Verb-with-no-complement** disappear
- **VP** → **Verb-with-NP-comp** NP prefer a morning flight
- **VP** → **Verb-with-S-comp** S said there were two flights

How would the rule for post-nominal relative clauses

```
RelClause → (who | that) VP
```

need to be modified if we wanted to deal properly with examples like “the earliest flight that you have”?

// “the earliest flight that you have” = you have the earliest flight
// NP VP-with-NP-comp NP

I take the earliest flight that you have.
```
<table>
<thead>
<tr>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
</tr>
<tr>
<td>NP</td>
</tr>
<tr>
<td>RelClause</td>
</tr>
</tbody>
</table>
```
that NP Verb-with-NP-comp

\[
S \rightarrow \text{NP } \text{VP}
\]

\[
\text{NP} \rightarrow \text{Det Nominal} \quad \text{// a morning flight}
\]

\[
\text{Nominal} \rightarrow \text{Noun} \quad \text{// flight}
\]

\[
\text{Nominal} \rightarrow \text{Noun Nominal} \quad \text{// morning flight}
\]

\[
\text{NP} \rightarrow \text{Proper-Noun}
\]

\[
\text{VP} \rightarrow \text{Verb-with-no-complement} \quad \text{// disappear}
\]

\[
\text{VP} \rightarrow \text{Verb-with-NP-comp} \text{ NP} \quad \text{// prefer a morning flight}
\]

\[
\text{VP} \rightarrow \text{Verb-with-S-comp} \text{ S} \quad \text{// said there were two flights}
\]

\[
\text{NP} \rightarrow \text{NP RelClause} \quad \text{// flight that …}
\]

\[
\text{RelClause} \rightarrow \text{RelClause-no-need-Obj}
\]

\[
\text{RelClause} \rightarrow \text{RelClause-need-Obj}
\]

\[
\text{RelClause-no-need-Obj} \rightarrow \text{(who | that) VP} \quad \text{// the earliest flight that disappear}
\]

\[
\text{// the earliest flight that has 100 passengers}
\]

\[
\text{RelClause-need-Obj} \rightarrow \text{(who | that) NP Verb-with-NP-comp} \quad \text{// the earliest flight that you have}
\]

Exercise 2. Bottom-up Parsing (0p- but plus 1point to students who did it correctly)
(Backtrack parsing with reduce-preference)

Exercise 3. (Exercise 10.2 - Speech and Language Processing). (3points)
Write an algorithm (in pseudo-code) for eliminating left-recursion based on the intuition on page 370 (Section 10.3 - Speech and Language Processing)

“Rewrite each rule of the form \( A \to A\beta \) according to the following schema, using a new symbol \( A' \):

\[
\begin{align*}
/A \to A\beta | \alpha & \Rightarrow A \to \alpha A' \\
A' \to \beta A' | \epsilon
\end{align*}
\]

function TOP-DOWN-PARSE(input, gramma) returns a parse tree

//from Speech and Language Processing
agenda ← (Initial S tree, Beginning of input)
current-search-state ← POP(agenda)
loop
  if SUCCESSFUL-PARSE?(current-search-state) then
    return TREE(current-search-state)
  else
    if CAT(NODE-TO-EXPAND(current-search-state)) is a POS then
      if CAT(NODE-TO-EXPAND(current-search-state)) in POS(CURRENT-INPUT(current-search-state)) then //\( A \to \alpha \)
        PUSH(APPLY-LEXICAL-RULE(current-search-state), agenda)
      else //\( A \to A\beta \)
        PUSH(APPLY-RULES(current-search-state, gramma), agenda)
    else
      if agenda is empty then
        return reject
      else
        current-search-state ← POP(agenda)
  end

// new function for eliminating left-recursion based
// suppose that rules in the grammar has been sorted so that \( A \to BC \) always stands before \( A \to \alpha \)

function REWRITE-RULE (grammar)

// new function for eliminating left-recursion based
aramle = new Rule();
gramma1 = new Grammar();
// I don’t want to use the same grammar since removing rules from the grammar will change index of the rules that are being considered
loop
  arule ← NEXT(grammar,arule);
  if LHS(arule) = FIRST(RHS(arule)) then //\( A \to A\beta \)
    // create a new symbol that hasn’t been used in grammar and gramma1
    A’ = CREATE-NONTERMINAL-SYMBOL(grammar,gramma1);
    arule1 = new Rule();
    loop // find rules that have LHS = A
      arule1 ← NEXT(grammar,arule1);
      if((LHS(arule1)=LHS(arule) and CAT(RHS(arule1)) is a terminal ) then //\( A \to \alpha \)
        newrule = CREATE-RULE(A,RHS(arule1)+A’) //\( A \to \alpha A’ \)
ADD-RULE(grammar1, newrule);
newrule = CREATE-RULE(A', TAIL(arule) + A');  // A' → βA'
ADD-RULE(grammar1, newrule);
newrule = CREATE-RULE(A', ε);  // A' → ε
ADD-RULE(grammar1, newrule);
REMOVE-RULE(grammar, arule);  // remove A → Aβ from grammar
  // I don’t want to remove A → α since the grammar may have B → AC
end loop
end loop
ADD-RULE-SET(grammar1, grammar);
return grammar1;

Exercise 4. Consider the following sentences: (4points)

List A
i. Joe is reading the book.
ii. Joe had won a letter.
iii. Joe has to win.
iv. Joe will have the letter.
v. The letter in the book was read.
vi. The letter must have been in the book by Joe.
vii. The man could have had one.

List B
i. *Joe has reading the book.
ii. *Joe had win.
iii. *Joe winning.
iv. *Joe will had the letter.
v. *The book was win by Joe.
vi. *Joe will can be mad.
vii. *The man can have having one.

a. Write a context-free grammar that accepts all the sentences in list A while rejecting the sentences in list B. (1point)

S → NP VP
NP → Det Noun       the letter, the man, the book
NP → Proper-Noun     Joe
NP → NP PP           the letter in the book
PP → IN NP           in the book
VP → bePresent + VBG  is reading
VP → bePast + VBG     was reading
VP → bePresent + VBN (past participle) is written
VP → bePast + VBG     was read
VP → havePresent + VBN(past participle) has won, have won
VP → havePast + VBN(past participle) had won
VP → havePresent + to + VB has to win
VP → havePast + to + VB had to win
bePresent → am|is|are
bePast → was|were
havePresent → has|have
havePast → had
VP → MD + have + VBN (past participle) must have been
VP → MD + have + VBN (past participle) could have had
VP → Verb
VP → VP NP
VP → VP PP           must have been in the book by Joe
b. Implement one of the chart-based parsing strategies (in pseudo-code) and, using the grammar specified in part (a), demonstrate that your parser correctly accepts all the sentences in A and rejects those in B. You should maintain enough information in each entry on the chart so that you can reconstruct the parse tree for each possible interpretation. Make sure your method of recording the structure is well documented and clearly demonstrated. (2points – 1p for pseudo-code, 1p for examples)

I use CKY algorithm. Since CKY only accepts binary branching rules, I have to write a function to convert trinary branching rules to binary branching rules. (You can write your own algorithm.)

CKY algorithm

- for i := 1 to n
  - Add to [i-1,i] all categories for the i\textsuperscript{th} word
- for width := 2 to n
  - for start := 0 to n-width
    - Define end := start + width
    - for mid := start+1 to end-1
      - for every constituent X in [start,mid]
        - for every constituent Y in [mid,end]
        - for all ways of combining X and Y (if any)
          - Add the resulting constituent to [start,end] if it’s not already there.

function \textsc{convert-rule} (grammar)
// convert trinary branching rules to binary branching rules
arule = new Rule();
gr\textsuperscript{1} = new Grammar();
loop
  arule = NEXT(grammar);
  if TRINARY(arule) then
    // the RHS of arule has 3 symbols: A \to BCD
    A’ = CREATE-NONTERMINAL-SYM BOL(grammar,grammar1);
    rule1 = CREATE_RULE(LHS(arule), FIRST(RHS(arule)) + A’);
    // rule 1: A \to BA’
    ADD_RULE(grammar1, rule1);
    rule2 = CREATE_RULE(A’, TAIL(RHS(A)) );
    // rule 2: A’ \to CD
    ADD_RULE(grammar1, rule2);
  else // if arule isn’t a trinary rule, then add arule to grammar
    ADD_RULE(grammar1, arule);
end loop
return grammar1;
Joe is reading the book.

<table>
<thead>
<tr>
<th></th>
<th>Joe</th>
<th>is</th>
<th>reading</th>
<th>the</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NP</td>
<td>Proper-Noun</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>VP</td>
<td>VB, bePresent</td>
<td>VP</td>
<td>VP</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>VBG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Det</td>
<td>NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Joe has reading the book.

<table>
<thead>
<tr>
<th></th>
<th>Joe</th>
<th>has</th>
<th>reading</th>
<th>the</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NP</td>
<td>Proper-Noun</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>VP</td>
<td>VB, havePresent</td>
<td>VP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>VBG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Det</td>
<td>NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There is no S for the whole sentence ⇒ “Joe has reading the book” cannot be recognized as a correct sentence with the current grammar.

c. List three (distinct) grammatical forms that would not be recognized by a parser implementing the grammar in part (a). Provide an example of your own for each of these grammatical forms. (1 point)

S → NP VP
NP → Det Nominal  // the book
NP → Proper-Noun  // Joe
VP → havePresent + been + VBG  // has been reading
VP → VP NP
havePresent → has|have
Example: Joe has been reading the book.

VP → MD + be + VBG  // will be walking
Example: Joe will be reading the book.

VP → MD + have been + VBG  // will have been reading
Example: Joe will have been reading the book.