Exercise 10.1

The following grammar generates sequences of round and square brackets, so that round brackets only can directly contain square brackets and square brackets only directly contain round brackets, e.g.

([][])()
[([])]
[][][()([])]

S is the start symbol.

S → (B) A
  | [A] B
A → (B) A
  | ε
B → [A] B
  | ε

a) Compute nullable, First sets and Follow sets for all non-terminal symbols.

	Nullable	First	Follow
S	false	([	$
A	true	(	)
B	true	[	]

First(S) = { (, [}
First(A) = { (}
First(B) = { [}
Follow(S) = {$}
Follow(A) = {] }
Follow(B) = {) }

b) Define the predictive parsing table.

	(	)	[	]	ε	$
S	S→(B)A	-	S→[A]B	-	-	-
A	A→(B)A	-	-	A→ε	-	-
B	-	B→ ε	B→[A]B	-	-	-

S → (B) A : (
S → [A] B : [
A → (B) A : (
A → ε :]
B → [A] B : [
B → ε :)

c) Is it a LL(1)-grammar?
yes

d) Show step-by-step, how a table driven top-down parser analyzes the input string

[()()].

stack input — next operation

→ (S, [()()]) — push start symbol S on stack

→ (S, [()()]) — S expand with S → [A] B
→ (B]A[, [()()]) — [/[consume symbol [
→ (B]A, ()()]) — A expand with A → (B) A
→ (B]A)B(, ()()]) — (/(consume symbol (
→ (B]A)B,)()]) — B/ε expand with B → ε
→ (B]A),)()]) —)/) consume symbol)
→ (B]A, ()]) — A expand with A → (B) A
→ (B]A)B(, ()]) — (/(consume symbol (
→ (B]A)B,)]) — B/ε expand with B → ε
→ (B]A),)]) —)/) consume symbol)
→ (B]A,]) — A/ε expand with A → ε
→ (B],]) —]/] consume symbol]
→ (B, ε) — B/ε expand with B → ε
→ (ε, ε)

Exercise 10.2 - obligatory (7 points)

The following context-free grammar with start symbol S is given:

S → BSA | cAB
A → ε | bA
B → aBAc | A

This grammar has the following properties.

	nullable	First	Follow
S.	false.	a, b, c.	$, b
A.	true.	b.	a, b, c, $
B.	true.	a, b.	a, b, c, $

First(S) = { a,b,c }
First(A) = { b }
First(B) = { a,b }
Follow(S) = { $,b }
Follow(A) = { a,b,c,$ }
Follow(B) = { a,b,c,$ }

a) Compute the predictive parsing table for this grammar.

	a	b	c	ε	$
S	S → BSA	S → BSA	S → BSA S → cAB	–	–
A	A → ε	A → ε A → bA	A → ε	–	A → ε
B	B → aDAc B → A	B → A	B → A	–	B → ε

S → B S A : c b a
S → c A B : c
A → epsilon : c b a $
A → b A : b
B → a B A c : a
B → A : c b a $

b) Is it a LL(1)-grammar?
Yes

Exercise 10.3 - obligatory (4 points)

The context-free grammar

S → cSa | AcB
A → aA | b
B → bB | ε

This grammar has the following properties.
First(S) = { a,b,c }
First(A) = { a, b }
First(B) = { b }
Follow(S) = { a }
Follow(A) = { c }
Follow(B) = { a }

has the following LL(1) predictive parsing table:

	a.	b.	c.	$
S.	S → AcB.	S → AcB.	S → cSa.	-
A.	A → aA.	A → b.	-	-
B.	B → ε.	B → bB.	-	B → ε

a) Show how a table-driven top-down parser analyzes the following input strings. Specify stack content and remaining input for each step.
(1) aabcb

stack input	next operation
→ (S, aabcb)	push start symbol S on stack
→ (S, aabcb)	S/a expand with S → AcB
→ (BcA, aabcb)	A/a expand with A → aA
→ (BcAa, aabcb)	a/a consume symbol a
→ (BcA, abcb)	A/a expand with A → aA
→ (BcAa, abcb)	a/a consume symbol a
→ (BcA, bcb)	A/b expand with A → b
→ (Bcb, bcb)	b/b consume symbol b
→ (Bc, cb)	c/c consume symbol c
→ (B, b)	B/b expand with B → bB
→ (Bb, b)	b/b consume symbol b

stack input	next operation
→ (B, ε)	B/ε expand with B → ε
→ (ε, ε)	

(2) cba

stack input	**next operation**
→ (S, cba)	push start symbol S on stack
→ (S, cba)	S/c expand with S → cSa
→ (aSc, cba)	c/c consume symbol c
→ (aS, ba)	S/b expand with S → AcB
→ (aBcA, ba)	A/b expand with A → b
→ (aBcb, ba)	b/b consume symbol b
→ (aBc, a)	c/a not possible.

Which of the strings belong to the language of the grammar?

Strings: a^+ bc b^+

b) Show a leftmost derivation of the string aabcb.

S → AcB (A → aA)
S → aAcB (A → aA)
S → aaAcB (A → aA)
S → aaAcB (A → b)
S → aabcB (B → bB)
S → aabcB (B → ε)
S → aabcb

Exercise 10.4 - obligatory (5 points)

The following grammar defines arithmetic expressions, consisting of numbers, operators + and *, and brackets. As usual, operation * has higher precedence than +, and both operators are handled as being left-associative.

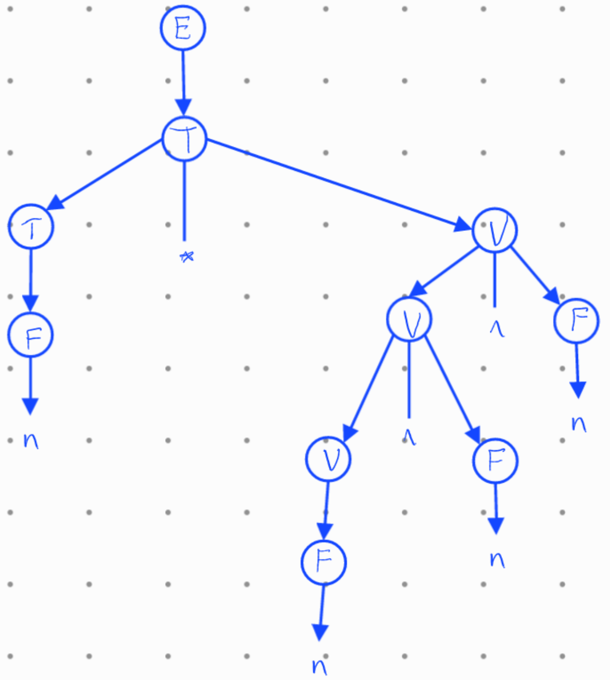
E → E + T
  | T
T → T * F
  | F
F → number
  | (E)

a) Augment the grammar with a power operator ^, where 3^5 denotes 3^5
. The power operator ^ should have higher precedence than multiplication * and it should be right-associative, e.g.
5*4^3^2
should be read as
5 * (4 ^ (3 ^ 2)).

E → E + T
  | T
T → T * V
  | V

T → V ^ F
| F
F → number | (E)

b) Draw the derivation tree for 5*4^3^2 according to the extended grammar (2, 3, 4, 5 are considered number symbols).



Exercise 10.5
The following grammar is not LL(1). Give three reasons why not (without computing the predictive parsing table).
S → AaS | AB
A → A+A | a
B → Bb | cc

Exercise 10.6
Eliminate left recursion from the following grammar:
S → SAa
| Bb
| Sc
A → aa
B → Bb
| bB
| cc

Exercise 10.7 - obligatory (4 points)
Given is the following excerpt from a grammar for expressions.
Expr → Expr + Term
| Expr -Term
| Term
a) Give two reasons why the grammar is not LL(1).
- left recursion on Expr → Expr + Term (a repetition of any number of "+ term")
- ambiguous

b) Transform the productions so that they are no longer left recursive.

Expr → (Term) (+ Term) (- Term)*

Expr → Term Expr1

Expr1 → + Term Expr1

| - Term Expr1

| Term

A → A + 7

| A - 7

A + 7

A + 7 + 7

7 + 7 + 7