european social fund in the czech republic
EUROPEAN UNION
MINISTRY OF EDUCATION, YOUTH AND SPORTS
OP Education for Competitiveness
INVESTMENTS IN EDUCATION DEVELOPMENT

1 Definite Clause Grammars (DCG)

Exercise 1.1: We need a grammar which recognizes/generates the language a^{2n} for $n > 0$. Analyse the behaviour of the following grammars.

```
1.
s --> s, [a,a].
s --> [].

2.
s --> [a,a].
s --> s, [a,a].

3.
s --> [a,a].
s --> [a,a], s.
```

What is the native Prolog representation of the correct grammar?

Solution 1.1:
Grammar no. 1 always loops (for both recognition and generation).
Grammar no. 2 can generate the language and accepts correctly the words which belong to the language. However, it loops for words that do not belong to the language.
Grammar no. 3 is correct for both recognition and generation.
Native representation of the correct version:

```
s([a, a|A], A).
s([a, a|A], B) :- s(A, B).
```

Exercise 1.2: Write a DC grammar for recognition/generation of the (context-sensitive) language $a^n b^n c^n$ for $n \geq 0$. The grammar should return as its argument an appropriate n for every word generated/recognized.

Solution 1.2:

```
abc(N) --> a(0,N), b(N), c(N).
a(N,N) --> [].
a(N,V) --> [a], {N1 is N + 1}, a(N1,V).
b(0) --> [].
b(N) --> [b], {N > 0, N1 is N - 1}, b(N1).
c(0) --> [].
c(N) --> [c], {N > 0, N1 is N - 1}, c(N1).
```

Exercise 1.3: Write a DC grammar for the recognition of correct arithmetic expressions in the postfix notation containing operators $+, -$ and nonnegative integers. (To be more simple, the grammar can also accept isolated nonnegative integers.)

For example, the grammar should recognize the expression $5\ 2 - 4\ 3\ 2 - + +$. Suppose that the expression is already represented as the appropriate list of terminals: `[5,2,'-',4,3,2,'-','+','+']`.

Extend the grammar so that it evaluates the recognised expression. Extend it further to return the parse tree as one of its arguments.

Solution 1.3: We can easily write a left-recursive grammar (that would not work in Prolog!):

```
e --> f.
e --> e, e, ['+'].
e --> e, e, ['-'].
f --> [X], {integer(X), X>=0}.
```

After elimination of left recursion we get the correct Prolog grammar:

```
e --> f, e1.
e1 --> [].
e1 --> e, ['+'], e1.
e1 --> e, ['-'], e1.
f --> [X], {integer(X), X>=0}.
```

Expression-evaluating extension of the grammar:

```
e(Y) --> f(X), e1(X,Y).
e1(X,Y) --> [], {X=Y}.
e1(X,Y) --> e(V), ['+'], {W is X+V}, e1(W,Y).
e1(X,Y) --> e(V), ['-'], {W is X-V}, e1(W,Y).
f(X) --> [X], {integer(X), X>=0}.
```

Further extension of the grammar (returns a parse tree):

```
e(T,Y) --> f(Z,X), e1(Z,T,X,Y).
e1(Z,T,X,Y) --> [], {Z=T,X=Y}.
e1(Z,T,X,Y) --> e(T1,V), ['+'], {W is X+V, T2=plus(Z,T1)}, e1(T2,T,W,Y).
e1(Z,T,X,Y) --> e(T1,V), ['-'], {W is X-V, T2=minus(Z,T1)}, e1(T2,T,W,Y).
f(Z,X) --> [X], {integer(X),X>=0,Z=leaf(X)}.
```