Programmation Avancée et Compilation 2001
(Solutions)

2 Lexical Analysis

Exercise 2

(a) In order to derive the languages produced by the two grammars, we have a look at some words produced by them.

1. From Aexp we can derive 0, 725, −1, −007, ...
2. From Bexp we can derive 0.0, 34.725, 126.0, . . ., but also simply a dot.

We can conclude that the first grammar produces the set of *integers* as its language, whereas the second grammar produces the set of *floating-point numbers* together with a dot.

(b) Alternatively, the grammars can be described by the following EBNF expressions:

1.	Aexpr	::=	("−" | Digit) {Digit}
2.	Bexpr	::=	{Digit} "." {Digit}
	Digit	::=	("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9")

Exercise 3

At http://lampwww.epfl.ch/courses/progavancee01/HandScanner.java, you now find a completed version of class HandScanner.

The function nextToken() uses nextCh() to read one input symbol after the other, and basically has to consider two cases: (1) blank spaces or new line, for instance, can simply be ignored, which can be implemented by the following **while**-loop:

```
while (ch == ' ' || ch == '\t' || ch == '\n' || ch == '\r') {
    ch = nextCh();
}
```

(2) Tokens have to be grouped according to the grammar and mapped to the non-terminals they are derived from. This can be implemented by a **switch**-expression terminating for EOF:

```
switch(ch) {
case -1: return EOF;
...
}
```

Digits are grouped to form integer literals:

```
case '0': case '1': case '2': case '3': case '4':
case '5': case '6': case '7': case '8': case '9':
    chars = "" + (char) ch;
    ch = NextCh();
    while (ch ≥ '0' && ch ≤ '9') {
        chars += (char) ch;
        ch = NextCh();
    }
    return INTLIT;
```

For the other terminal symbols considered in the grammar, the nonterminal can be returned immediately; if some other symbol is read, and ERROR message is returned:

```
case '(':  ch = nextCh();
           return LPAREN;
case ')':  ch = nextCh();
           return RPAREN;
case '+':  ch = nextCh();
           return PLUS;
case '−':  ch = nextCh();
           return MINUS;
case '∗':  ch = nextCh();
           return TIMES;
case '/':  ch = nextCh();
           return DIV;
default:   ch = nextCh();
           return ERROR;
```