level	expression forms	explanation
15	literal	literal (integer, float, string, boolean)
15	identifier	variable
15	`f(e,...,e)`	function call
14	`v++`, `v--`	post-increment, post-decrement
13	`++v`, `--v`	pre-increment, pre-decrement
13	`-e`	numeric negation
12	`e*e`, `e/e`	multiplication, division
11	`e+e`, `e-e`	addition, subtraction
9	`e<e`, `e>e`, `e>=e`, `e<=e`	comparison
8	`e==e`, `e!=e`	(in)equality
4	`e&&e`	conjunction
3	`e||e`	disjunction
2	`v=e`	assignment

The table is straightforward to translate to a set of BNFC rules. On the level of literals, integers and floats ("doubles") are provided by BNFC, whereas the boolean literals `true` and `false` are defined by special rules.

```
EInt.    Exp15  ::= Integer ;
EDouble. Exp15  ::= Double ;
EString. Exp15  ::= String ;
ETrue.   Exp15  ::= "true" ;
EFalse.  Exp15  ::= "false" ;
EId.     Exp15  ::= Id ;

ECall.   Exp15  ::= Id "(" [Exp] ")" ;

EPIncr.  Exp14  ::= Exp15 "++" ;
EPDecr.  Exp14  ::= Exp15 "--" ;

EIncr.   Exp13  ::= "++" Exp14 ;
EDecr.   Exp13  ::= "--" Exp14 ;
ENeg.    Exp13  ::= "-" Exp14 ;

EMul.    Exp12  ::= Exp12 "*"  Exp13 ;
EDiv.    Exp12  ::= Exp12 "/"  Exp13 ;
EAdd.    Exp11  ::= Exp11 "+"  Exp12 ;
ESub.    Exp11  ::= Exp11 "-"  Exp12 ;
ELt.     Exp9   ::= Exp9  "<"  Exp10 ;
EGt.     Exp9   ::= Exp9  ">"  Exp10 ;
ELEq.    Exp9   ::= Exp9  "<=" Exp10 ;
EGEq.    Exp9   ::= Exp9  ">=" Exp10 ;
```

```
EEq.    Exp8  ::= Exp8  "==" Exp9 ;
ENEq.   Exp8  ::= Exp8  "!=" Exp9 ;
EAnd.   Exp4  ::= Exp4  "&&" Exp5 ;
EOr.    Exp3  ::= Exp3  "||" Exp4 ;
EAss.   Exp2  ::= Exp3 "=" Exp2 ;
```

Finally, we need a `coercions` rule to specify the highest precedence level, and a rule to form function argument lists.

```
coercions Exp 15 ;
separator Exp "," ;
```

- *The available types are* `bool`, `double`, `int`, `string`, and `void`.

```
Tbool.   Type ::= "bool" ;
Tdouble. Type ::= "double" ;
Tint.    Type ::= "int" ;
Tstring. Type ::= "string" ;
Tvoid.   Type ::= "void" ;
```

- *An identifier is a letter followed by a list of letters, digits, and underscores.*

Here we cannot use the built-in `Ident` type of BNFC, because apostrophes (') are not permitted! But we can define our identifiers easily by a regular expression:

```
token Id (letter (letter | digit | '_')*) ;
```

Alternatively, we could write

```
position token Id (letter (letter | digit | '_')*) ;
```

to remember the source code positions of identifiers.

The reader is advised to copy all the rules of this section into a file and try this out in BNFC, with various programs as input. The grammar is also available on the book web page.