

1. Конкретный синтаксис.

$\langle ident \rangle ::= \langle nondigit \rangle$
 $\quad | \quad \langle nondigit \rangle \langle ident_rest \rangle$
 $\langle ident_rest \rangle ::= \langle alphanum \rangle | \langle alphanum \rangle \langle ident_rest \rangle$
 $\langle cmp-op \rangle ::= '=' | '/=' | '<=' | '<' | '>' | '>=' \quad \langle md-op \rangle ::= '*' | '/' \quad \langle pm-op \rangle ::= '+' | '-'$
 $\langle unop \rangle ::= '-' | '!'$
 $\langle num-lit \rangle ::= \text{number}$
 $\langle bool-lit \rangle ::= 'T' | 'F'$
 $\langle lit \rangle ::= \langle num-lit \rangle | \langle bool-lit \rangle$
 $\langle expr \rangle ::= \langle binop-expr \rangle$
 $\langle binop-expr \rangle ::= \langle or-expr \rangle$
 $\langle or-expr \rangle ::= \langle and-expr \rangle '||' \langle or-expr \rangle | \langle and-expr \rangle$
 $\langle and-expr \rangle ::= \langle cmp-expr \rangle '&&' \langle and-expr \rangle | \langle cmp-expr \rangle$
 $\langle cmp-expr \rangle ::= \langle pm-expr \rangle \langle cmp-op \rangle \langle pm-expr \rangle | \langle pm-expr \rangle$
 $\langle pm-expr \rangle ::= \langle pm-expr \rangle \langle pm-op \rangle \langle md-expr \rangle | \langle md-expr \rangle$
 $\langle md-expr \rangle ::= \langle md-expr \rangle \langle md-op \rangle \langle pow-expr \rangle | \langle pow-expr \rangle$
 $\langle pow-expr \rangle ::= \langle unop-expr \rangle '^' \langle pow-expr \rangle | \langle unop-expr \rangle$
 $\langle unop-expr \rangle ::= \langle un-op \rangle \langle lit \rangle$
 $\quad | \quad \langle un-op \rangle \langle var \rangle$
 $\quad | \quad \langle un-op \rangle '(' \langle atom-expr \rangle ')'$
 $\langle atom-expr \rangle ::= \langle ident \rangle$
 $\quad | \quad \langle lit \rangle$
 $\quad | \quad \langle app \rangle$
 $\quad | \quad \text{'if' } \langle expr \rangle \text{'then' } \langle expr \rangle \text{'else' } \langle expr \rangle$
 $\quad | \quad \text{'let' var '=' } \langle expr \rangle \text{'in' } \langle expr \rangle$
 $\quad | \quad \langle expr \rangle \langle binop \rangle \langle expr \rangle$
 $\quad | \quad \langle unop \rangle \langle expr \rangle$
 $\quad | \quad '(' \langle expr \rangle ')'$
 $\langle app \rangle ::= \langle ident \rangle ' ' \langle ident \rangle$
 $\quad | \quad \langle ident \rangle ' ' \langle lit \rangle$
 $\quad | \quad \langle ident \rangle ' ' '(' \langle expr \rangle ')'$
 $\quad | \quad '(' \langle expr \rangle ') ' \langle ident \rangle$
 $\quad | \quad '(' \langle expr \rangle ') ' \langle lit \rangle$
 $\quad | \quad '(' \langle expr \rangle ') ' '(' \langle expr \rangle ')'$
 $\langle bind \rangle ::= \langle ident \rangle \langle arg \rangle '=' \langle expr \rangle$
 $\quad | \quad \langle ident \rangle '=' \langle expr \rangle$
 $\langle arg \rangle ::= \langle ident \rangle | \langle ident \rangle ' ' \langle arg \rangle$
 $\langle decl \rangle ::= \langle bind \rangle ';' \langle decl \rangle | \langle bind \rangle ';'$

Примеры.

1. Объявление функции/переменной.

```
f x = x;
n = 10;
```

2. Использование условного оператора `if`

```
f x = if x == 0 then 10 else 20 + x;
```

3. Использование `let`-связывания.

```
f y = let x = 10 * y in x ^ x;
```

4. Вызов функции

```
f x = x;
g = f 10;
```

```
ff x y = x + y;
gg = ff 10 20;
```

2. Абстрактный синтаксис

Представлен в виде АСД.

```
type Ast = [Decl]
```

```
data Decl = BindDecl Bind
```

```
data Bind = Bind Name [Name] Expr
```

```
data Expr = Var Name
          | Lit Lit
          | App Expr Expr
          | If Expr Expr Expr
          | Let Name Expr Expr
          | UnOp UnOperator Expr
          | BinOp BinOperator Expr Expr
```

```
data Lit = ILit Integer | BLit Bool
```