1. Конкретный синтаксис.

⟨*ident*⟩ ::= ⟨*nondigit*⟩
  | ⟨*nondigit*⟩ ⟨*ident_rest*⟩

⟨*ident_rest*⟩ ::= ⟨*alphanum*⟩ | ⟨*alphanum*⟩ ⟨*ident_rest*⟩

⟨*cmp-op*⟩ ::= ‘==’ | ‘/= ’ | ‘<=’ | ‘<’ | ‘>’ | ‘>=’ ⟨*md-op*⟩ ::= ‘*’ | ‘/’ ⟨*pm-op*⟩ ::= ‘+’ | ‘.-’

⟨*unop*⟩ ::= ’-’ | ’!’

⟨*num-lit*⟩ ::= number

⟨*bool-lit*⟩ ::= ’T’ | ’F’

⟨*lit*⟩ ::= ⟨*num-lit*⟩ | ⟨*bool-lit*⟩

⟨*expr*⟩ ::= ⟨*binop-expr*⟩

⟨*binop-expr*⟩ ::= ⟨*or-expr*⟩

⟨*or-expr*⟩ ::= ⟨*and-expr*⟩ ‘||’ ⟨*or-expr*⟩ | ⟨*and-expr*⟩

⟨*and-expr*⟩ ::= ⟨*cmp-expr*⟩ ‘&&’ ⟨*and-expr*⟩ | ⟨*cmp-expr*⟩

⟨*cmp-expr*⟩ ::= ⟨*pm-expr*⟩ ⟨*cmp-op*⟩ ⟨*pm-expr*⟩ | ⟨*pm-expr*⟩

⟨*pm-expr*⟩ ::= ⟨*pm-expr*⟩ ⟨*pm-op*⟩ ⟨*md-expr*⟩ | ⟨*md-expr*⟩

⟨*md-expr*⟩ ::= ⟨*md-expr*⟩ ⟨*md-op*⟩ ⟨*pow-expr*⟩ | ⟨*pow-expr*⟩

⟨*pow-expr*⟩ ::= ⟨*unop-expr*⟩ ‘^’ ⟨*pow-expr*⟩ | ⟨*unop-expr*⟩

⟨*unop-expr*⟩ ::= ⟨*un-op*⟩ ⟨*lit*⟩
  | ⟨*un-op*⟩ ⟨*var*⟩
  | ⟨*un-op*⟩ ‘(’ ⟨*atom-expr*⟩ ‘)’

⟨*atom-expr*⟩ ::= ⟨*ident*⟩
  | ⟨*lit*⟩
  | ⟨*app*⟩
  | ‘if’ ⟨*expr*⟩ ‘then’ ⟨*expr*⟩ ‘else’ ⟨*expr*⟩
  | ‘let’ var ‘=’ ⟨*expr*⟩ ‘in’ ⟨*expr*⟩
  | ⟨*expr*⟩ ⟨*binop*⟩ ⟨*expr*⟩
  | ⟨*unop*⟩ ⟨*expr*⟩
  | ‘(’ ⟨*expr*⟩ ‘)’

⟨*app*⟩ ::= ⟨*ident*⟩ ‘ ’ ⟨*app-args*⟩
  | ‘(’ ⟨*expr*⟩ ‘)’ ‘ ’ ⟨*app-args*⟩

⟨*app-args*⟩ ::= ⟨*app-arg*⟩ | ⟨*app-arg*⟩ ‘ ’ ⟨*app-args*⟩

⟨*app-arg*⟩ ::= ⟨*lit*⟩ | ⟨*ident*⟩ | ‘(’ ⟨*expr*⟩ ‘)’

⟨*bind*⟩ ::= ⟨*ident*⟩ ⟨*arg*⟩ ’=’ ⟨*expr*⟩
  | ⟨*ident*⟩ ’=’ ⟨*expr*⟩

⟨*arg*⟩ ::= ⟨*ident*⟩ | ⟨*ident*⟩ ’ ’ ⟨*arg*⟩

⟨*decl*⟩ ::= ⟨*bind*⟩ ’;’ ⟨*decl*⟩ | ⟨*bind*⟩ ’;’

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
```