Intro:

val x : array[5] = [8, 5, 12, 12, 15] → arrays
                    0 1 2   3   4

val i : [0..4] = 4 → range types
val elem : Int = x[i] → dynamic checks when needed
val l : Int = x.length

Lexer:

'[' , ']' , keywords: 'array', 'length'

']' ↳ RBRACKET()

'[' ↳ LBRACKET()

Parser:

Rules:

```
Type ::= array [IntLit] | [IntLit . . IntLit]
Expr ::= ID.length | ID[Expr]
Literal ::= [IntLit ArrayElems]
ArrayElems ::= epsilon | , IntLit ArrayElems
```

Name Analysis:

- add a few new things
- no significant change

Type checking:

typing rules:

$$\frac{a \text{ is an arrayLit. of size } m}{\Gamma \vdash a : \text{Array}(m)}$$

- Range and Int interchangable

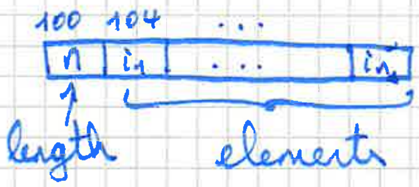
$$\frac{\exists n : \Gamma \vdash a : \text{Array}(n)}{\Gamma \vdash a.\text{length} : \text{Int}}$$

$$\frac{\exists n : \Gamma \vdash a : \text{Array}(n) \qquad \Gamma \vdash i : \text{Int}}{\Gamma \vdash a[i] : \text{Int}}$$

→ no checks for the correct changes yet.

Code Generation:

- store arrays in memory
  and return a pointer



- a[e]:
  - compute e
  - store in local
  - compare with length of array (0 ≤ ... < length)
  - if both hold: return the element
  - else: throw error
- a.length:
  - return length
- How to know when to check and when not to?
  - use env: map[Identifiers, Type] like in Typechecker.

```
if e matches:
    IntLiteral -> compare with length
                  during compiling
    Var        -> look in env if it is a correct range
    else       -> put check.
```