Types & Declarations

Types:

property associated with a name / variable that allows you to derive some other properties, typically:

- the way operators operates on them (or not)

  int + int
  float + float } different ways on how '+' operator works

- the way functions operates on them (or not)

  foo (int, int)

- storage details, e.g. size, width of a variable in memory.

  volatile
  private } behavior of variables
  static
  ....

Built in Types:

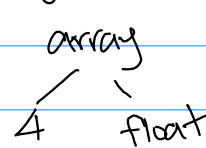
refer as primitive → int, float, char, bool, void, long int, ...

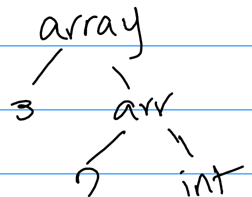
Compound Types:

Type constructors:

array (size, Type)
record (Type)

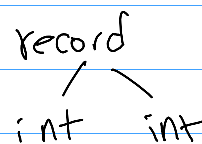
Type Expressions





Record: a list of types

range (Type)
list (Type)



record (int, int)

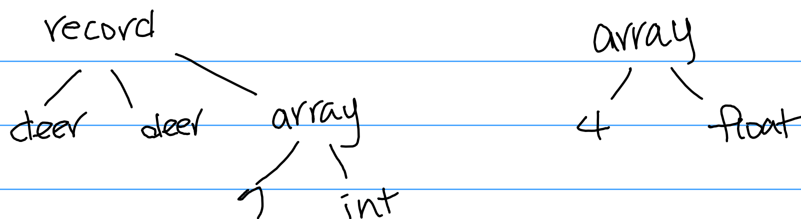
Types 2

name Types:

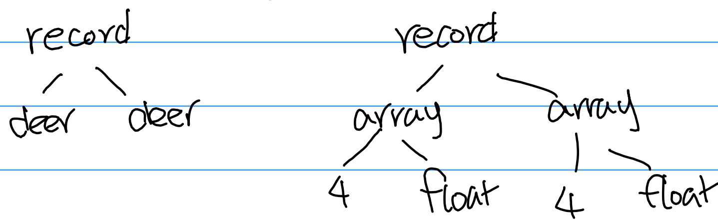
name stands in for a Type expression

'typedef' name type-expression

'deer' = array of size 4 of float



* When are two types are equivalent?



1. Structural Equivalence

two types are equivalent if:

a. same primitive type

b. same type constructors applied to equivalent types

c. one type is a name for the other type.

Recursive Type

moose = record

(record → int, moose)

②. Name Equivalence

Only options ⓐ and ⓑ

```
class duck {              char goose {
    int x;       ⟺            int x;
    int y;                     int y;
}                          }
```

Structural equivalence is the same, but not name equivalence

translate (point, dx, dy);   (point: record, dx: int, dy: int)

int * int → bool ≡ (int, int) → bool