A Type is a Set of Values

Consider the statement:

```c
int n;
```

Here we declare `n` to be a variable of type `int`; what we mean, `n` can take on any value from the set of all integer values.

Also observe that the elements in a type share a common representation: each element is encoded in the same way (float, double, char, etc.)

Also, all elements of a type share the same operations the language supports for them.
Def: A type is a set of values.

Def: A primitive type is a type programmer can use but not define.

Def: A constructed type is a user defined type.

Example: Java, primitive type

```java
float q;
```

q is of type float, only a value that is a member of the set of all floating point values can be assigned to q.

type float ⇒ set of all possible floating point values
Example: ML, primitive type

- val p = ...;

untyped variable → can assume a value of any type.

- val p:real = ...;
- 

Now p only accepts a value that is the member of the type real.
Example: Java, constructed type

```java
class Foobar { int i; String s; };

Foobar c = new Foobar();
```

Now the variable `c` only accepts values that are members of type `Foobar`;

object instantiations of class `Foobar`. 
**Example**: ML, constructed type

- `type foobar = int * string;`
- `val c:foobar = (1, "two");`

an element of type `foobar`. 
Types

Example: C, constructed type

```c
int a[3];
```

the variable `a` will accept values which are arrays of 3 integers.

e.g.: `int a[3] = {1,2,3};`
`int a[3] = {7,24,9}`

Example: ML, constructed type

```ml
- val L : int list = ...;
```

L will accept values which are integer lists – more formally, L will accept values that are members of type ‘int list’.
**Def:** a *subtype* is a *subset* of the elements of a type.

**Example:** Java

Short is a subtype of int: \( \text{short} \subset \text{int} \)

**Observations:**
(1) converting a value of a subtype to a values of the super-type is called *widening* type conversion. (safe)
(2) converting a value of a supertype to a value of a subtype is called *narrowing* type conversion. (not safe)

**Example:** Java

\( \text{float} \subset \text{double} \)
C, C++, and ML treat functions as just another data type that can be manipulated

- Functions can be passed as values; just as values that belong to other data types
- Functions belong to function types

Example: in ML consider the function type

```
real → int
```

This type represents the set of all functions from real to int.

We have seen some members of this type:

- floor: real → int
- ceil: real → int
- round: real → int
Function Types

Example: Functions as values
- fun myfun (x:real):int = round(x);
val myfun = fn:real -> int

- val foo = myfun;
val foo = fn:real -> int

- foo(3.4);
val it = 3 : int

Example: Functions as function arguments
- fun myfun(f:real -> int) = …;
- myfun(round);
- myfun(ceil);

☞ A function is just an element of a particular function set.
Why do we use types?

- Types allow the computer/language system to assist the developer write better programs. **Type mismatches** in a program usually indicate some sort of **programming error**.
  - **Static type checking** – check the types of all statements and expressions at compile time.
  - **Dynamic type checking** – check the types at runtime.
I. **Name Equivalence** – two objects are of the same type of and only if they share the same *type name*.

**Example: Java**

```java
Class Foobar {
    int i;
    float f;
}
```

```java
Class Goobar {
    int i;
    float f;
}
```

```java
Foobar o = new Goobar();
```

Error; even though the types look the same, their names are different, therefore, Java will raise an error.

➔ **Java uses name equivalence**
II. Structural Equivalence – two objects are of the same type if and only if they share the same type structure.

Example: ML
- type person = int * int * string * string;
- type mytuple = int * int * string * string;
- val joe: `person = (38, 185, "married", "pilot"):mytuple;

Even though the type names are different, ML correctly recognizes this statement.

ML uses structural equivalence.
Exercises

- Describe the type associated with the set of values \{-1,-2,-3,-4,\ldots\}, call this type Q.
- Describe the type associated with the set of values \{-2,-4,-6,-8,\ldots\}, call this type P.
- Is there a subtype-supertype relationship between this types? If so, what is it?
- Let x be a variable of type Q and y be a variable of type P, then is the assignment \( x := y \) a safe assignment? Why? Why not?
- Describe the type associated with set Q \( \rightarrow \) P.
- Types are sets of values, typically with a common representation and common set of operations.
- Types in programming languages allows compilers and interpreters to check for consistency in your programs.
- Inconsistencies usually show up as type mismatches.
- Type equivalence between constructed types can be established in one of two ways, name equivalence or structural equivalence.
Assignment # 5 – see website