# 14 Types and Type Checking

Wednesday, October 30, 2024 12:05 PM

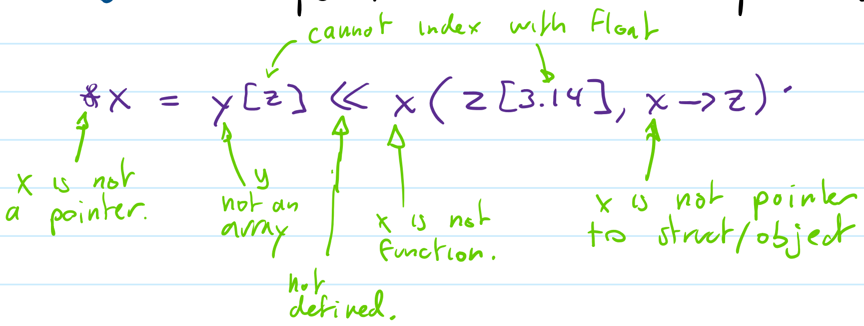
- SEMANTICS

  the study of meaning.

e.g C++

```
float x, y, z;
*x = y[z] << x(z[3.14], x->z);
```

follow the syntax rules.
but
it is meaningless.
because operators don't match operands.




In programming languages, semantics is concerned about identifiers and their correct use.

- TYPE CHECKING:

  ¿Are named entities used in a matter consistent with their definition?

  ie:
  - using a variable with appropiate operators.
  - call a function with appropiate number and types of parameters.
  - members of compound types are used correctly

  ¿How?

  - The compiler/parser needs to remember the declaration of every named entity

- THE SYMBOL TABLE:

A table of named entities and their attributes.

| names | attributes |
|---|---|
| x | int |

examples of symbol table entries

- variable

  float y; y : float

- constant

  const int z = 3; z : int, 3, const.

- type

  ```
  struct cell {
     int row, col;
  }
  ```

  cell: struct, 2, int row, int col

- function

  ```
  void foo (int r, int c)
  {
     ≡
  }
  ```

  foo: function, void, 2, int r, int c,

Entries are different in different programming languages

C++:

```
int z[3];
float foo ( int x, bool y, char& c);
```

| name | attributes |
|---|---|
| z | int array |
| foo | function 3 (int, bool, char&) "float" |

Python

```
z = [1,2,3]
def foo ( x, y, c ) :
  ........
```

| name | attr |
|---|---|
| z | [1,2,3] |
| foo | function 3 |

Pascal

```
z : ARRAY [10..12] OF INTEGER;
FUNCTION foo ( x : INTEGER,
               y : BOOL,
               VAR c : char ) : REAL
```

| | |
|---|---|
| z | ARRAY, INT, 10..12 |
| foo | function 3 Integer Bool VAR char. :REAL. |

C++ Consequences.

```
int foo ( string y );
int foo ( char* y );
string& foo ( string y );
```

foo(string)

foo(char*)

⊗

```
string& foo ( string y );
```
⊗

## Pascals enumerated Arrays:

```
days = {Monday, Tuesday, Wednesday, Thursday, Friday}
z : ARRAY [Monday..Friday] OF INTEGER;
```

- TYPE CONVERSION:

Some languages will automatically apply type conversion:

e.g. a + x     a: float<br>
                   x: int

most parsers will identify the need for conversion and automatically convert.

int → float<br>
float → int.

¿What conversions are Implicit?

C++ flexible<br>
Python runs and maybe crash.<br>
Pascal Strict.

E.g. C++ flexibility

int ↔ float ↔ char ↔ bool<br>
(int: all sizes; float: all-sizes)

```
foo ( Cat c )
{
  ⋮
}

foo(3);
```

```
class Cat
{
  ⋮
  void meaw( );
  ⋮
  Cat ( int n )
  { ⋮
  }
}
```

becomes an implicit conversion unless specified explicit.

E.g Pascal is strict.

C                 Pascal.

E.g. Pascal is strict.

**C**

```
int x = 70;
char c = '!';

while ( 'W' - x ) {
  c = '!' + x;
  print("%c", c)
  x = x + 1;
}
```

('W' is char, x is int, 'W' - x is int; c is char, '!' is char, x is int)

{ 0 is false. everything else is true

**Pascal.**

```
VAR x : INTEGER = 70;
    c : CHAR = '!';

WHILE  ORD('W') - x > 0 DO BEGIN

   c := CHR( ORD('!') + x );

END;
```

explicit conversions.

- TYPE EQUIVALENCE:

a = b

¿When are two entities of the same type?

1) Name type equivalence:

- two entities are of the same type if they are declared using the same type name.

Dog bob;  Cat tom;  Dog fido;

Just look at text.

1) Structural type equivalence:

- two entities are of the same type if they share the same internal structure.

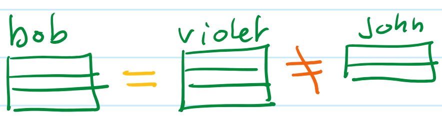
Imagine:

$C^2$

```
struct pnt        struct color      struct cell
{                 {                 {
  int x,y,z;        int r,g,b;        int r,c;
}                 }                 }
```

Under structural type equivalence, which are of the same type?

pnt bob
color violet
cell john




bob = violet;

- Hard to implement:
  - you need to test structure

- Hard to implement:
  - you need to test structure
  - structure can be nested.

- STRONG vs WEAK TYPING:

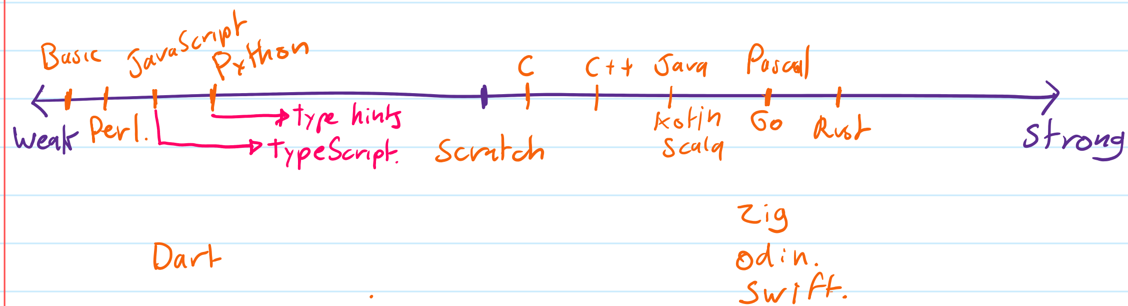
A characterization of Programming Languages

"Strongly" typed if

- type violations are detected at compile/parse time.
- type conversions are expicit
- the type of a named variable remains fixed.

General idea:

Detect errors at compile time,
instead of letting them happen at runtime.




EOF