14 Types and Type Checking

Monday, October 23, 2023 11:01 AM

- Semantics:
  the study of meaning.

  "Colorless green ideas sleep furiously"
  Adj Adj N V Adv.
  meaningless

  "A private cloud can be untrustworthy"
  meaningless before.-

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

  In programing languages, Semantics is mostly about identifiers and their use.

- TYPE CHECKING:
  - ¿ Are named entities used in a matter consistent with their definition?
    i.e: • using a variable with appropiate operators
    • calling a function with appropiate number and types of parameters.
  - ¿ How?
    the compiler/parser needs to remember declarations of entities in a program.

    THE SYMBOL TABLE
    A table of named entities and their attributes.

names	attributes
x	int

what is stored in the symbol table.

variable:
float x — x : float

constant
const int y=3 — y : const, int, 3

types:
struct cell {int r,c} — cell : struct, 2, int r, int c

function
void foo(int r, int c) — foo : function, 2, int r, int c, void.

different in different languages:

C++

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

name	attribute
z	int array
foo	function, 3, int, bool, char&

Python

```
z = 3
def foo ( x, y, c ) : …
```

z	3
foo	function, 3

Pascal

```
z : ARRAY [10..13] OF INTEGER;
FUNCTION bar ( x : INTEGER; y : BOOLEAN ) : REAL
```

z	int, 10, 13
bar	function, 2

x : int
y : boolean
Real.
return type.

- TYPE CONVERSION

Some languages will automatically convert types when allowed

e.g $a + x$ a: float, x: int

most parsers will identify an automatic implicit conversion here

int → float.

Implicit Conversions:

C++ : Flexible

Python : run and maybe crash.

Pascal : strict.

foo(Dog d)

class Dog:
    Dog(int n)

foo(3):

E.6.

C++

```
int x = 70; char c = '!';
while ('W' - x) {
    c = x;
    cout << c;
    x++;
}
```

'W' - x : int

c = x; : char = int

0 is false, everything else is true.

ghijk ....

Pascal.

explicit char to int

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

(ORD('W') - x) : int

int → bool

```
c = CHR( ORD(c) + x )
```

ORD(c) : char to int

x : int

ORD(c) + x : int

explicit int to char.

c ←

(ghijk ....)

explicit int to char.
c ←

- TYPE EQUIVALENCE:

a = b.

When are two entities of the same type?

1) Name type equivalence

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

pop x; chop y; pop z;

easy:- just look at text.

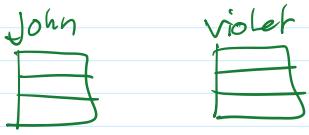
2) Structural type equivalence

- two types are equal if they have the same internal structure.

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

```
pnt john;
color violet;

john = violet;
```



- Hard:- - you need to test structure
  - Structure could be nested.

- STRONG vs WEAK TYPING:

A characterization of Programming Languages.

Strongly typed if

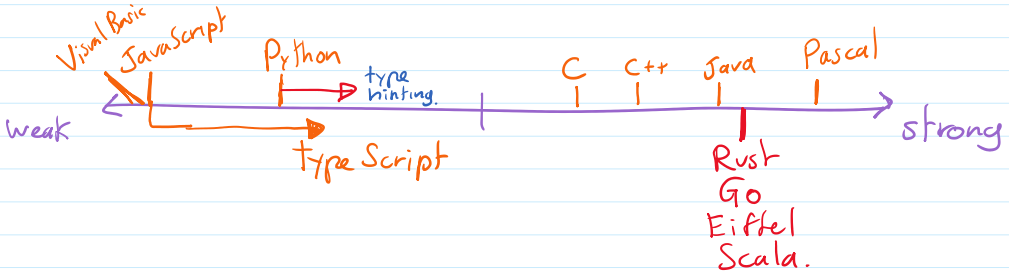
- type violations are detected at compile time.
- type conversions are explicit.
- the type of a named entity remains fixed.

General idea:

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

General idea:

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



—EOF—