Nominal vs. Structural Type Systems

Reminder: Types

- Often broken down into
  - Elementary (primitive, atomic types)
    - C types: int, char, double, float...
  - Compound : generated using *type constructors* – operators that accepts types and generates a new type.
    - C constructs: struct, enum, union,arrays, pointers...

Definitions

- *Nominal equivalence:* two types are equivalent if they have the same name in the same context
- *Structural equivalence:* two types are equivalent if they have the same structure

Example

```
struct {
  int x;
  int y;
} a1;
```

```
struct {
  int x;
  int y;
} b1;
```

```
a1 := b1;
```

// legal in structural languages

// illegal in nominal languages

Pascal as a Pure Nominal Language

```
a: array[1..100] of Integer;
b: array[1..100] of Integer;

a := b; // illegal!
```

- Which parameters can be passed to the following procedure?

```
procedure sort(var a: array[1..100] of integer)
begin
    ...
end
```

C as a Hybrid Language

- `structs` (and unions) are nominal...

```
typedef struct {int i;} S1;
typedef struct {int i;} S2;

S1 s1;
S2 *p = &s1; //error!
```

- All other type constructors create structural types

```
typedef int A1[100];
typedef int A2[100];

A1 a1;
A2 *p = &a1; //OK
```

Intra-program Communication and Nominal Typing

- File types in Pascal are nominally typed

```
Type
fREC = file of Record
  id: Integer;
  name: array[1..100] of Char;
end;
Var
  fr1: fREC;
```

  - Hence, a file type definition is valid only within the scope in which it is declared
  - A communication of two modules through files violates the nominal typing rules…

Intra-program Communication in C

- It is possible to declare the same type in different modules
  - Either by #including a common header file
  - Or by manually defining the type twice
- The compiler views these definitions as the same type
  - Even if they are different.
  - The reason: C’s type system is weakly typed.