TYPE SYSTEMS

Type - set of values and operations that can be applied on those values

Typesystem → Basic Types (int, char, float ...)
→ Typeconstructors (struct ..)
→ Type inference (variable type not explicitly specified
→ Type compatibility (which assignments are allowed ?)

⇓

Principally about type equivalence

int a, float b
a = b ?

↓

Goal: To see what is legal & what is not.

NAME EQUIVALENCE (N.E.)

Name of the datatype should be the same.

Eg1
```
typedef int cm;
typedef int inch;

inch x;
cm y;
```
x = y → Not name equivalent as name of x is inch and y is cm.

Eg2
```
int x ; int y ;
```
x = y → name eq as name of x = y = int

Eg3
```
int x, y ;
```
x = y → Name eq as name of x, y = int

Eg4
```
int *x, int *y
```
x = y Not name equivalent as pointers are anonymous data type.

because there is no explicit data type called pointer to int.

Egs

```
int a[0...4];        or    int a[4], b[4];
int b[0...4]
```

a=b, not Name equivalent.

a=b → not name equivalent as array is anonymous datatype

↓

To make it name equivalent

```
typedef array int[0..4] foo;
```

```
foo a, b;
```

↓

a=b → name equivalent as a and b have name foo.

INTERNAL NAME EQUIVALENCE (I.N.E)

If the program interpreter gives the same internal name to two different variables, then they are internally name equivalent.

```
int a[4], b[4];
int c[4];
```

a=b → not Name eq,

→ but Internal Name eq, [Internally converted to same name as they are declared on same line]

a=c → not name equivalent

→ not Internal name equivalent.

Internal name equivalence → more expressive, Less restricted | than Name Equivalence.

But!! a and c also can be type compatible, right ??

Don't you think we can do a=c....? Let's see if this is possible in Structural equivalence!! :)

STRUCTURAL EQUIVALENCE (S.E)

Rule 1. same builtin types are S.E

Rule 2. Pointers to S.E types are S.E

Eg1:
```
typedef int cm;
typedef int inch;
cm x;
inch y;
```
x = y ✓ Structually Equivalent

Eg2:
```
int *a;
float *b;
```
a = b ?
- ① same type (i.e) a,b are ptrs ⇒ follows Rule 1
- ② but point to different S.E types ⇒ doesn't follow rule 2

Hence not S.E.

Rule 3: Determining STRUCT structural equivalence

→ Two structure are S.E iff

St1 { x_1 : T_1 , x_2 : T_2 , ... x_k : T_k }
(variable, Type)

St2 { y_1 : P_1 , y_2 : P_2 , ... y_k : P_k }
(variable, Type)

→ St1 and St2 are S.E iff

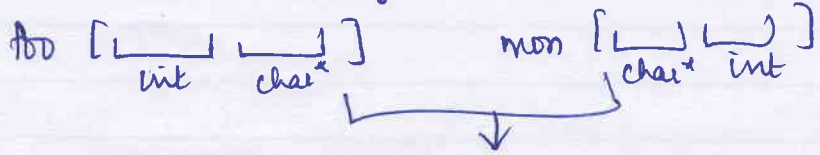
T_1 is S.E to P_1
T_2 is S.E to P_2
⋮
T_k is S.E to P_k

→ variable names don't matter, the order of the types matter.

Eg3

```
struct foo {          struct mon {          struct nike {
    int a;                string b;             int c;
    string b;             int a;                string e;}
}                     }
```

foo = bar → Not S.E as they are stored in chunks internally as



order is different, hence we can't assign one to another.

mon = nike → S.E !!

Rule 4: S.E of arrays

→ Two arrays

a: type$_1$ a[i_1 ---- i_k]

b: type$_2$ b[j_1 ---- j_k]

a,b are S.E iff a,b have ① same no of dimensions
② same no of entries in each dimension
and
③ type1 and type 2 are S.E.

Rule 5: Two functions are S.E iff

ret_type1 f1 (type$_1$ a_1, type$_2$ a_2, ---- type$_n$ a_n)

ret_type2 f2 (type$_a$ b_1, type$_b$ b_2, ---- type$_k$ b_n)

f1, f2 are S.E iff for all parameters type$_1$ $\overset{S.E}{=}$ type$_a$, ---- type$_n$ $\overset{S.E}{=}$ type$_k$
[all parameters, same order, are S.E types]

ret_types are S.E ret_type1 $\overset{S.E}{=}$ ret type$_2$

note a $\overset{S.E}{=}$ b ⇒ a,b are S.E

GOAL :
For every pair of types in the program, see if they are S.E

→ fairly simple ...!!
Keep applying the 5 rules till base case is reached but there are some recursive cases where we can get quite confused and lose marks in exams also :'(.. Let's see a procedural way to determine S.E.

Eg: A recursive case

```
struct T1 {
    int a;
    T2* p; };

struct T2 {
    int b;
    T1* q; };
```

→ To see if T_1 and T_2 are S.E

member 1: int & int are S.E ✓

member 2: are T_2^* and T_1^* S.E

↓

we know that two pointers are S.E if they point to SE type

Hence T_2^* and T_1^* are S.E if T_2 and T_1 are S.E

loop!!

This can't be solved by simply applying our rules till base case. So, lets use

Structural equivalence algorithm

① Assume all types in program are S E as we don't know the otherwise
② create nxn table where n is number of types in program and each entry in the table is true if types are S.E & false otherwise
③ Initialize all entries to true

Repeat filling the table, till table doesn't changed for two successive iterations n & n+1.

→ check each entry in the table, and if T_i and T_j are not s.t, then set entry (i,j) in table to false

T_i = type of i

Eg: struct T_1 {
int a;
T_2 * p; };

struct T_2 {
int c;
T_3 * p; }

struct T_3 {
float a;
T_1 * p; }

Itr 0

	T1	T2	T3
T1	T	T	T
T2	T	T	T
T3	T	T	T

Itr 1

	T1	T2	T3
T1	T	T	F
T2		T	F
T3			T

Itr 2

	T1	T2	T3
T1	T	F	F
T2		T	F
T3			T

Itr 3 = Itr 2 Stop

	T1	T2	T3
T1	T	F	F
T2		T	F
T3			T

Trick: Enough to fill i,j as j,i is same. also diagonals are always true as T_i and T_i are always s.t. Also, fill the table from L to R using the entries of i,j from previous iteration.

Itr 1 eg: T1 and T2

int, int → s.t

T_2* and T_3* are s.t if T_2 and T_3 are s.t. In Itr 0, T_2, T_3 = T

⇒ T_1 and T_2 are s.t.

Itr 1

T1, T3

int, float → not s.t

Itr 1

T2, T3

int, float, not s.t

Itr 2

T1 and T2

int, int → s.t

T_2*, T_3* not s.t as T_2, T_3 = F in Itr 1

Eg:

T_1 : string

T_2 : ptr to T_1 (T_1^*)

T_3 : ptr to string ($string^a$)

Variables

T_2 x;

Assume α type to global table Str^α y, z;

T_3 w;

Q1) x = w

N.E → No as name of x, w are different

I.N.E → No as I.N's are diff

S.E → Yes from table

Q2) y = w → anonymous type → no name

N.E → No name of y, w are diff

I.N.E → No

S.E → Yes

Q3) y = z

N.E → No name of y, z are diff as ptrs are anonymous data type

I.N.E → Yes as they are declared in same line, internally at same loc.

S.E → Yes → Table

Itr0

	T1	T2	T3	α
T1	T	T	T	T
T2		T	T	T
T3			T	T
α			T	T

Itr1

	T1	T2	T3	α
T1	T	F	F	F
T2		T	T	T
T3			T	T
α				T

Itr2 = Itr1

T1, T2, string, ptr, not same types, hence not S.E

T2, T3 → ptr to T1, ptr to string → ptrs to S.E types as T_1 → S.E to string.

Important;

Mostly,