Names and Scope


```
{
  int x=1;
  if (1==1) {
    int x=2;
  }
  print(x);
}
```

C-Language: 1 is printed — Why?
Java refuses to compile — Why?
Java script: 2 is printed — Why?

] Different scoping rules

Uses latest scope

Sub-scope

What if a language allows nested scope?
What " " " offer a global scope?

Mapping of names:

```
int x=3;
int y=4;
int z=x+y;
```

Referencing environment:

x — int — 3
y — int — 4
z — int — 7

Visibility determined by the program structure:

```
int x=1;
if (1==1) {
  int x=2;
}
```

Referencing environment:

x	int	1

Sub-scope →

x	int	2

x	int	1

Static scope derives the visibility by the program structure

Names and Scope

Dynamic scope determines the visibility by the call reference.

bash (Bourne-Again Shell)

```
x=1
function f() {
  echo $x;
}
function g() {
  local x=2;
  f;
}
g
```

x	1

stack

main
g
f

x	2

Multiple Scopes

Standard ML

```
type t = int;
val t = 17;
structure t =
struct
  val x = 23;
end
val x: t = t + t.x;
```

Scope for
- type names
- variable names
- structure names

x	1

Type Alias
Variable
Submodule

Scope govern the visibility of bindings
Reference environments store bindings, map names to attributes
Both form of I allow nesting

Names and Scope

Static bindings
- Language design time, e.g. Reserved keywords
- Language implementation time, e.g. Bit allocation to different types, stack and heap size
- Compile-time, e.g. Associating constant values with variables, Functionss in same file.
- Link-time, e.g. Function call that calls a function in another file.

Dynamic bindings
- Load-time, e.g. Assignment of physical memory.
- Run-time, e.g. Allocation in heap

Memory assignment
Relative to start maybe

Static — Fast, Determined at Compile-time
Dynamic — Slow but Flexible, Determined at Run-time