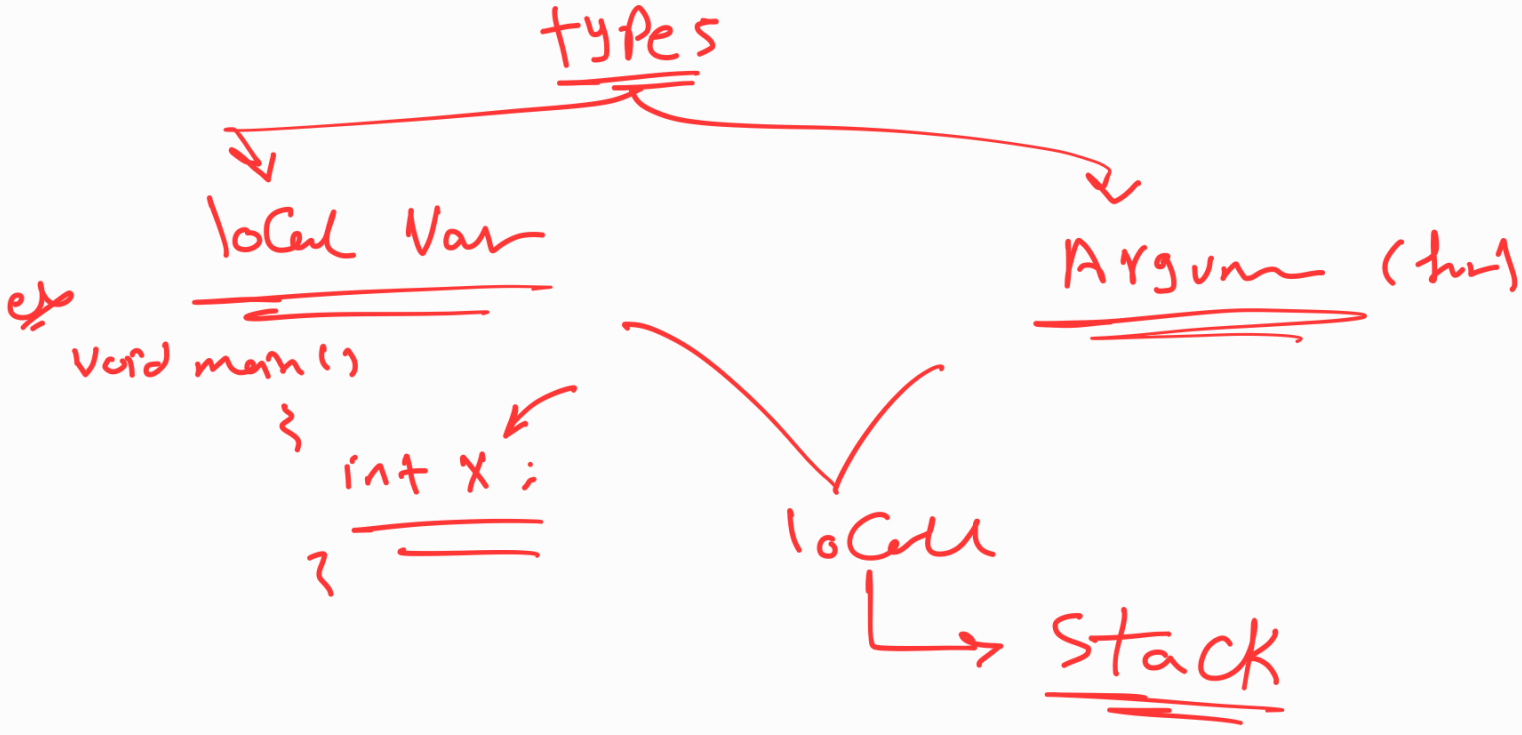
Local Var vs Global Var

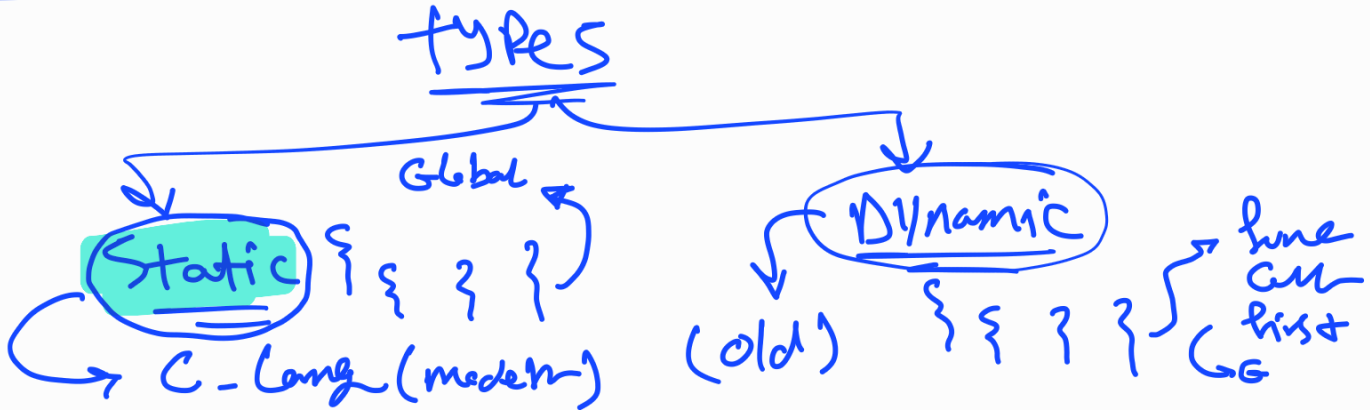
1] Local Var

↳ defined in any fun



- Scope
- Lifetime
- mem-Seg
- Init- Value

Scope



Local Var

- Scope ⟹ { } within Block
- Lifetime ⟹ within fun
- mem seg ⟹ Stack
- Init- Value ⟹ Garbage Value

```
int x ;
```

hint

→ local ≡ Automatic Var

---

[2] Global Var

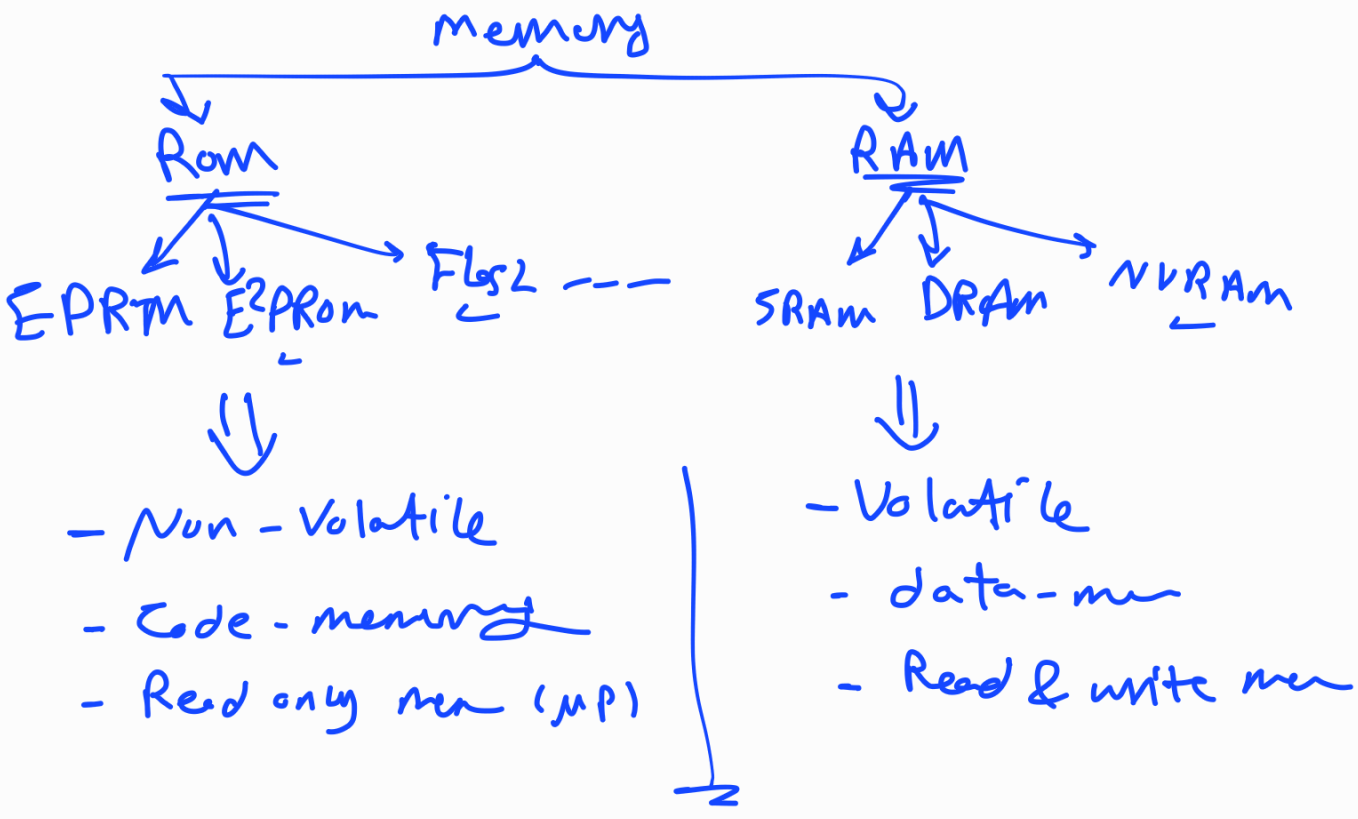
→ defined outside any fun

- Scope ⟹ all the program
- lifetime ⟹ Run time

```
int x = 1 ;
int x = 0 ;   int x ;
```

- mem seg
  - .data ⟹ initi- (≠0)
  - .bss ⟹ Not initi- or ini- (=0)
- int- Value → (=0)

Memory layout



- Stack → Local Var
- data → .data →
  Seg → .bss →
- heap → Dyn — Alloc —
- .text → Code

⇒ Recursion

⇒ It is a situation happens when a function calls it self

- Directly
- "In-directly"

```
void func()
{
    ====
    func();
    ====
}
Void main()
{
    ====
    func();
}
```

main → frame

(Stackoverflow)

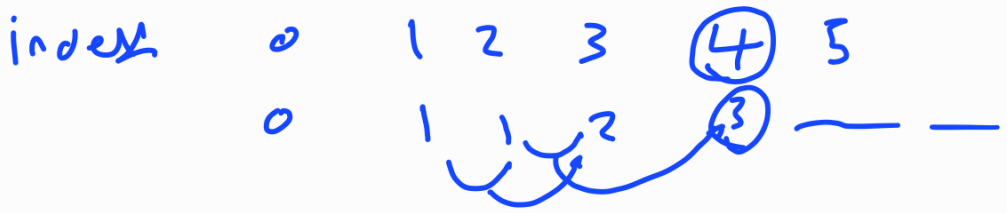
main → f1 → f2
f1 ← f2

ex ⇒ factorial

$n! = n \cdot (n-1)(n-2) \cdots 1$

```
int fact(int num)
{
    if (num <= 1)
    {
        return 1;
    }
    else {
        return (num * fact(num-1));
    }
}
```

fact ✗



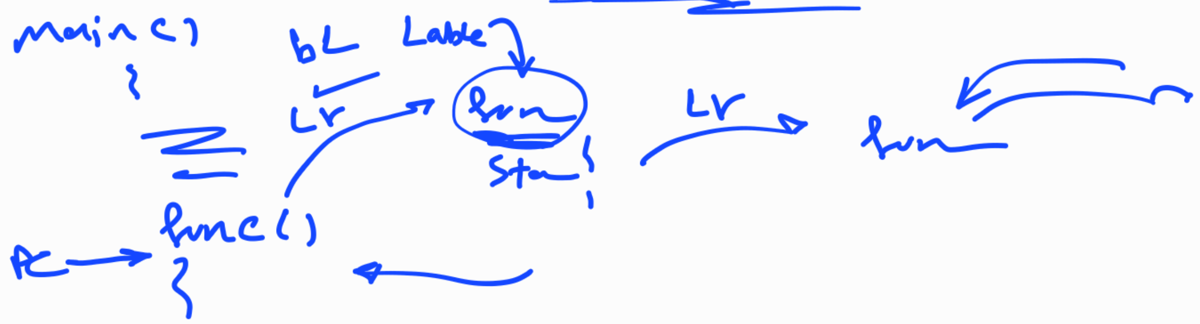
Recursion

- Adv
  - simple code
- dis Adv
  - ① infinite loop
    - Stack ov-f
  - ② Non-determinism

infinite loop

- Normal Loop
  - while(1);
  - for(;;);
  - No ov-f
- (Recursive Loop)
  - fun
  - ov-f (sure)

* Context Switch



⇒ switch Context

1. suspend the current seq
2. Record the return Addr
   - ↳ LR
     - ↳ load reg
3. Transfer Control to the fun
   - ↳ Jump ⇒ BL
4. excute the fun
5. use the recorded return Addr to go bake & resum the suspend code ↳ bX

two inst

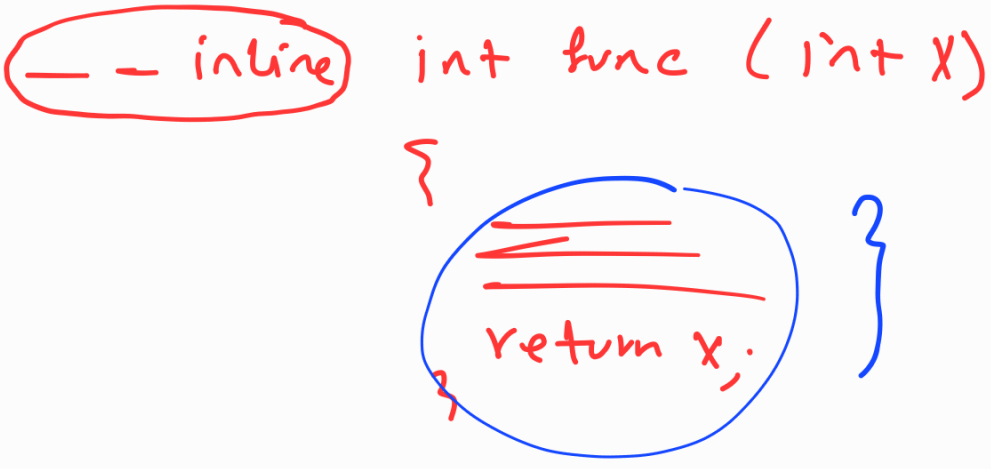
- BL ✓ — BL Lable → func name
- BX ✓ — BX LR

⇒ INline func

→ text repla ⇒ (Compiler)

Calling time (↓)

→ ANCI C (C 89)



→ C99 inline

Adv → exec time (↓)

dis-Adv

└→ Code Size (↑)

└→ (text)

memory vs time