Introduction about PL

System Component

- ① Processor
- ② Memory
- ③ I/o Peripheral

PL
- Communication between humans and devices to sent the instruction

PL Classification

Chain

Chain

- level → low | mid | High level
- execution →
  - Compiled
  - Interpirated
- Code style
  - oop
  - Pradual Programming.
- usage

---

① Machine language (1, 0)

- 0 → 0 volt
- 1 → 3.3v, 5volt

devices

operand

5+3

1010 001 0011 →

op Code

② Assembly language

Add
store
load

intel	amd	AVR	ARM
=	Add	sum	
Add	save	write	
store			

③ C_language

develop

5+3

Introduction of C_language

1978 Dennis Ritch & Brian kernighan

↓

Unix

→ C_Programming k&R

1989 → ANSI → C89

1990 → ISO → C90

1995 → ISO → C95

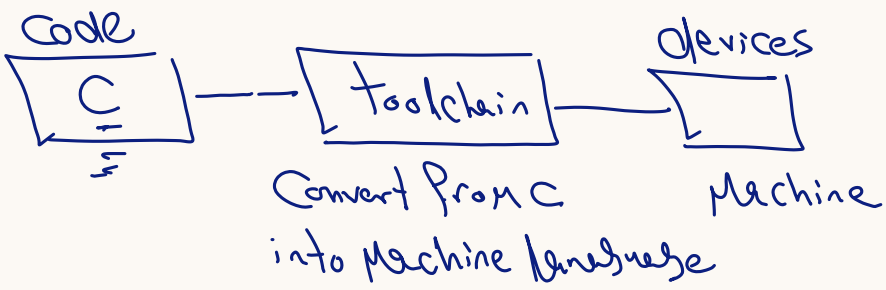
1999 → ISO → C99

2011 → ISO → C11

C89, C90, C95, C99 → E.S

ISO → Mostra C

C_Specs



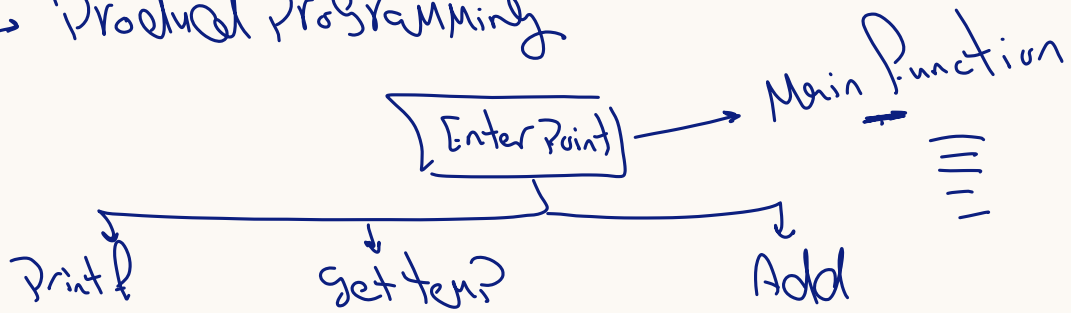
↳ Mid level

↳ Compiled →

↳ Convert code to Machine (build) →

↳ execution for output (Run) →

↳ Procedural Programming



Advantage

↳ efficiency & speed execution

↳ Full control on H.W

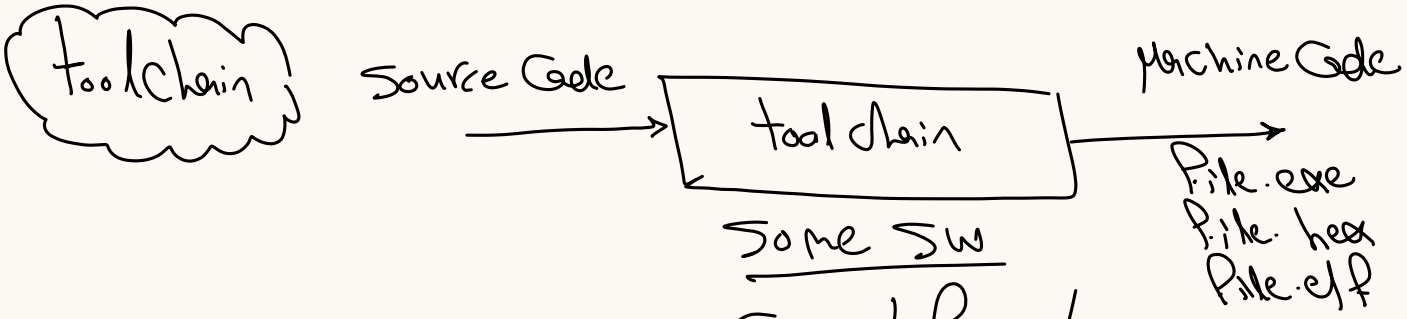
First Program

① need text editor + toolchain

② create file → extenstion file .c

```
#include <stdio.h>

int main ()
{
  Printf ("HappyHacking");

  return 0;
}
```

file1.c



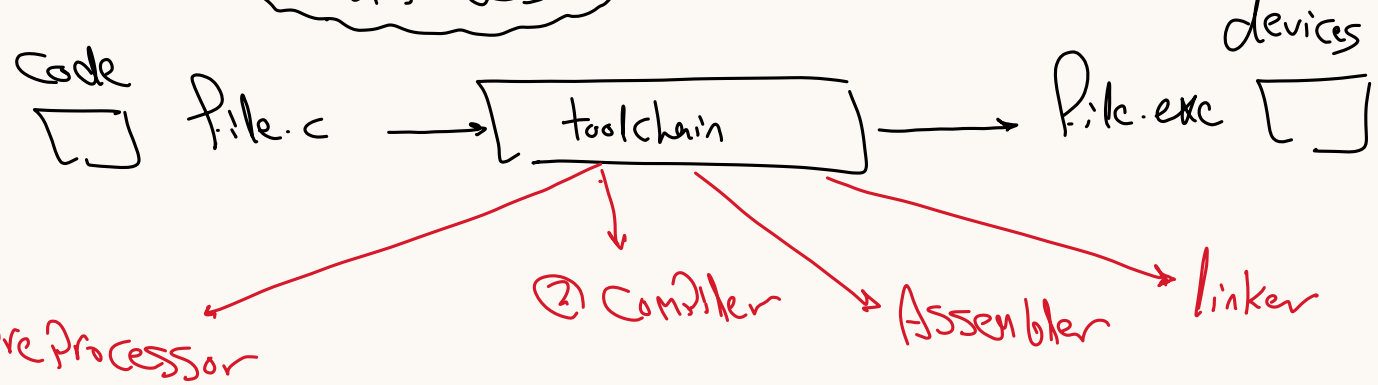
Some SW

Convert from low language into Machine language

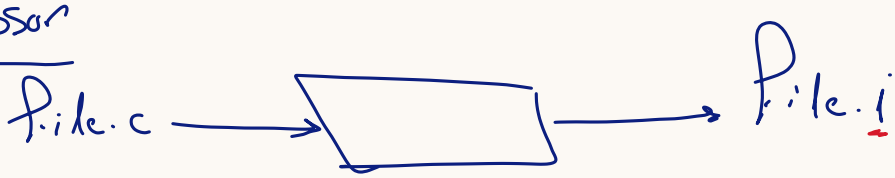
C_toolchain

- ① gcc toolchain
- ② Turbo C toolchain
- ③ Clang C toolchain

toolchain for C Build process



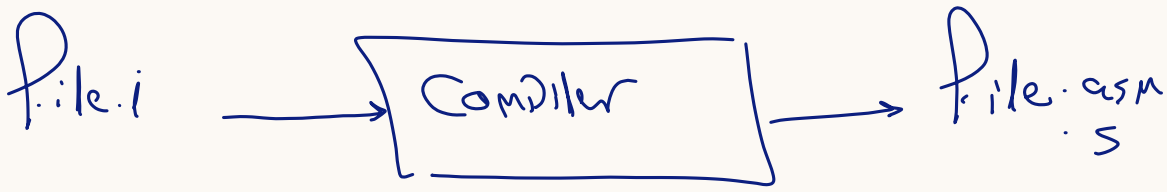
[1] PreProcessor



Text Replacement for PreProcessor directives

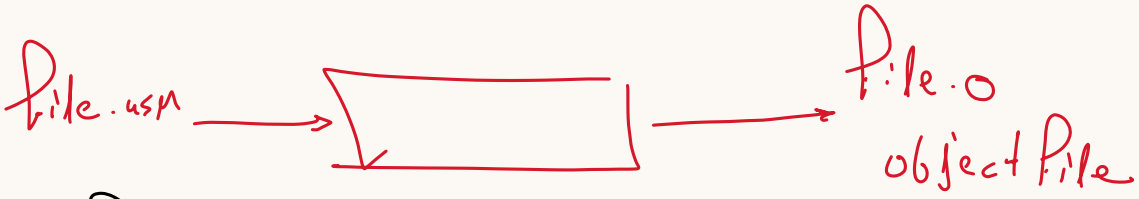
#include #define #undef #if #elif #else
#endif #error #warning #ifndef #ifdef

[2] Compiler



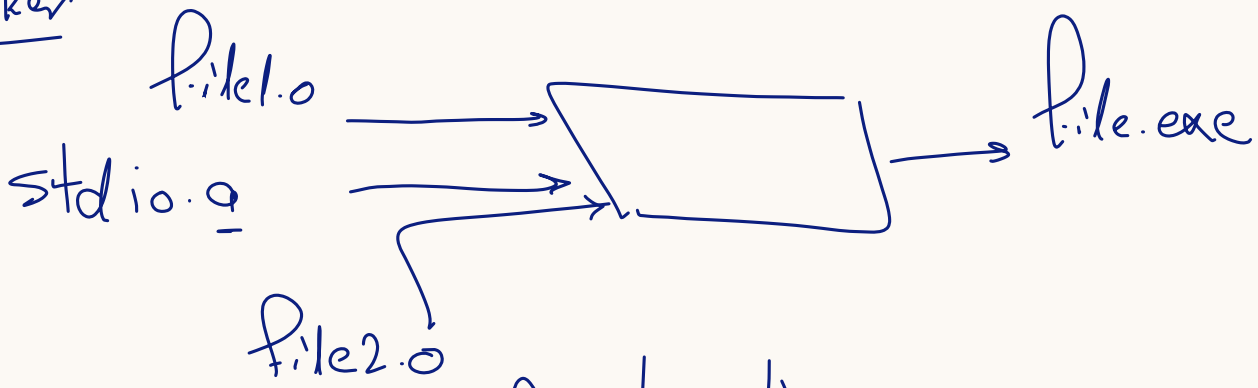
① check error
② optimization
③ Convert from c into Assembly specific

[3] Assembler

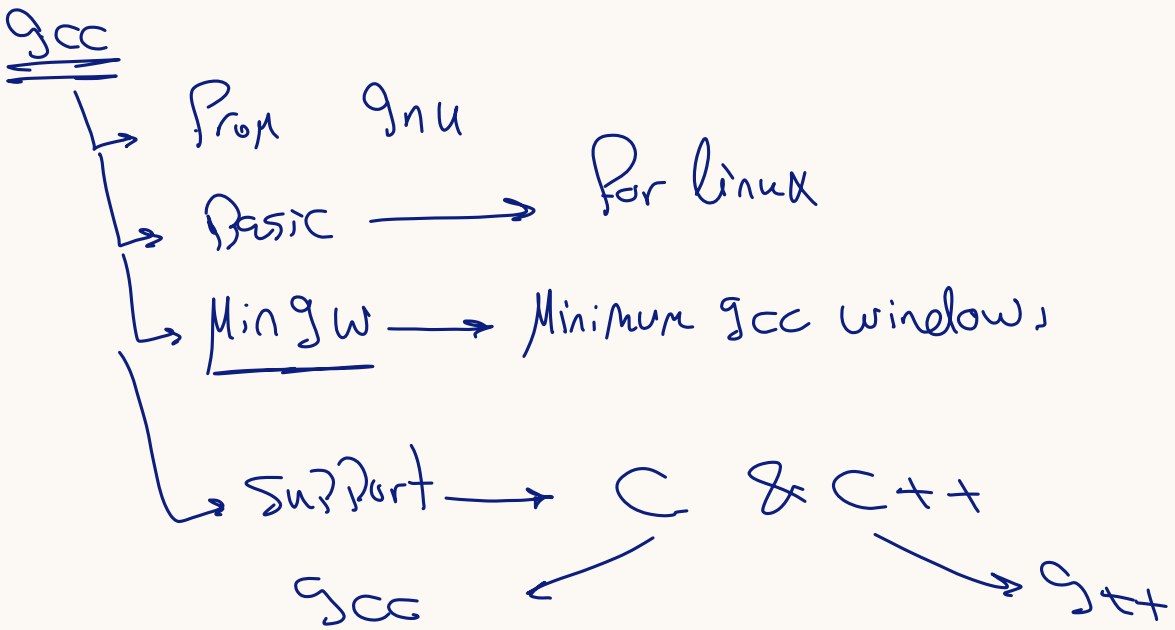


Convert from Assembly into Binary language

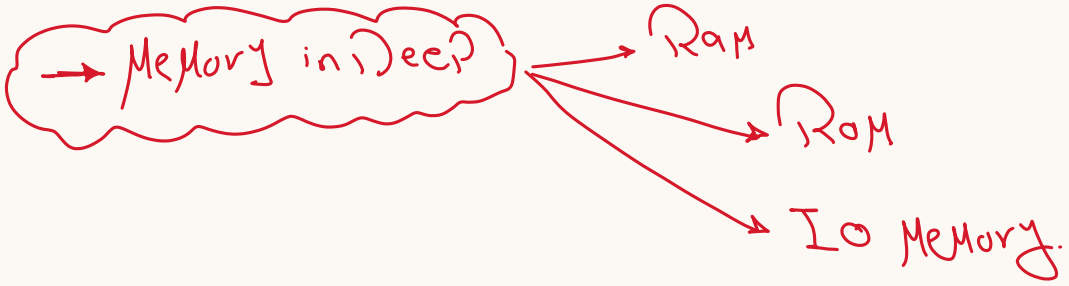
[4] linker



- Link all objects file together
- Phasical Address

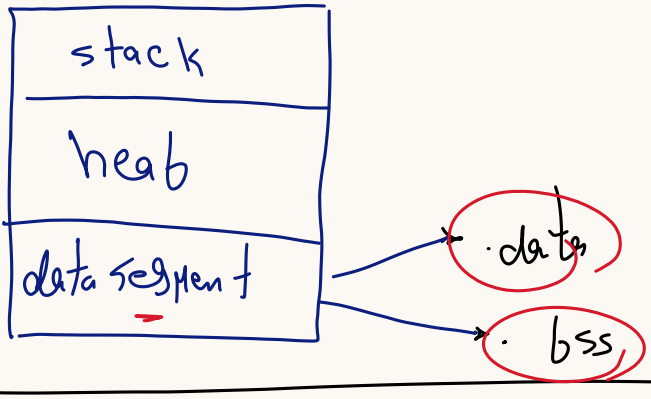


* Variables & Modifiers & keyword



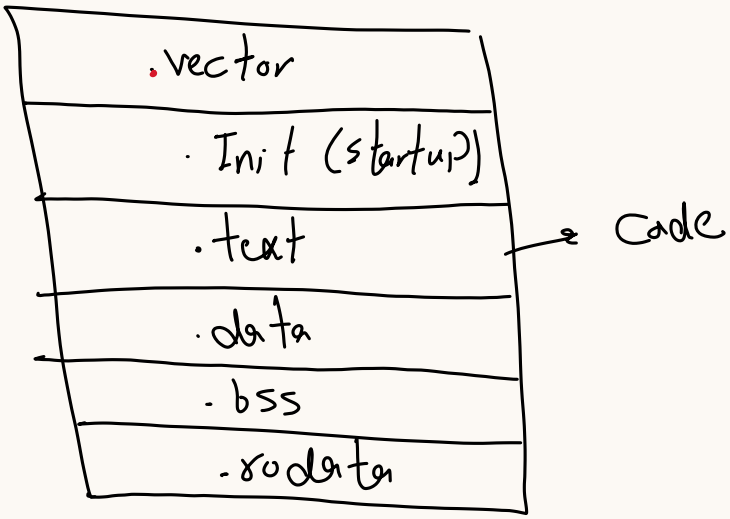
Ram: Random Access Memory

* store Data During Runtime
* Called Data Memory
* Processor → R/W
*



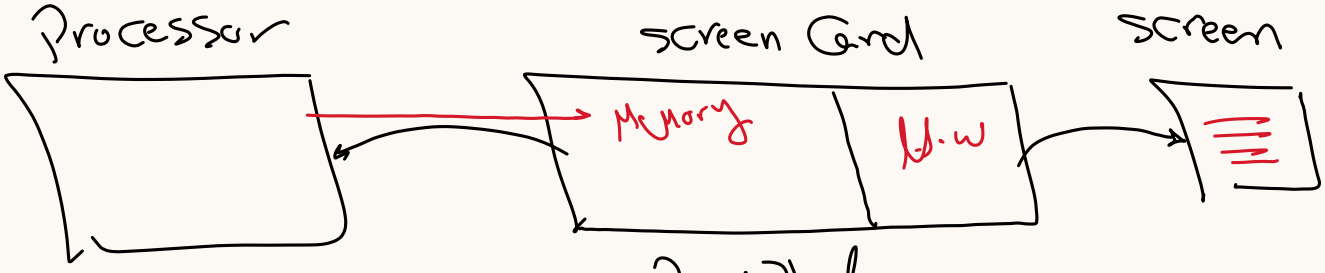
Rom: Read only Memory

↳ Processer → Readonly
↳ store for code before Run time
↳ Called Code Memory
↳



* IO Memory

* Processor Communicate with memory only.



Periphel
- H.w circuit
- Memory

Notes
- unit Memory → bit → 0, 1
- Byte → 8 bit
-

Variable
- create var → store Place in Memory (allocate)

*static allocate →

* Definition

DataType varName = InialValue;

allocate Place in Memory with Inial value

* Declartion

DataType varName;

allocate Place in Memory without value

Inial value	Garbage value

Mosra C → Best Case create var with inial value
↳ if you don't know value start with zero

Data type

↳ Decrmitve

Alph + Number ↳ char → at least 1B-Byte

Number ↳ int → at least 2B-Byte | 4B-Byte | 8B-Byte

Floating ↳ Float → at least 4B-Byte

Floating ↳ double → at least 8B-Byte

↳ void → Nothing → Function

* Naming Rule

① No Spaces : float gas_Sensor ;

② Not start Number

③ Not start symbol except underscore

int _gasSensor ; char #_No1 ;

④ Not use keyword

⑤ Max size for Name = 32 Char

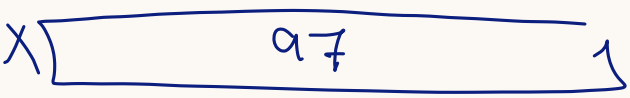
int Number1 = 5;
Definition
Assume 4B-Byte

char X = 20;
Definition
1Byte

Float tmp;
Declartion
4B-Byte

char z = 97;

Char x = 'A' ; → 97



Compiler ASCII Code

Scope & life time

*static allocate

*Block scope

and var create inside Block

Called **local var**

Block mean inside { }

Function scope

For scope

if scope

```
#include <stdio.h>
int y = 30;
int main ()
{
  char z = 20;

return 0;
}
```

*any var create outside any Block

Called **global var**

Memory

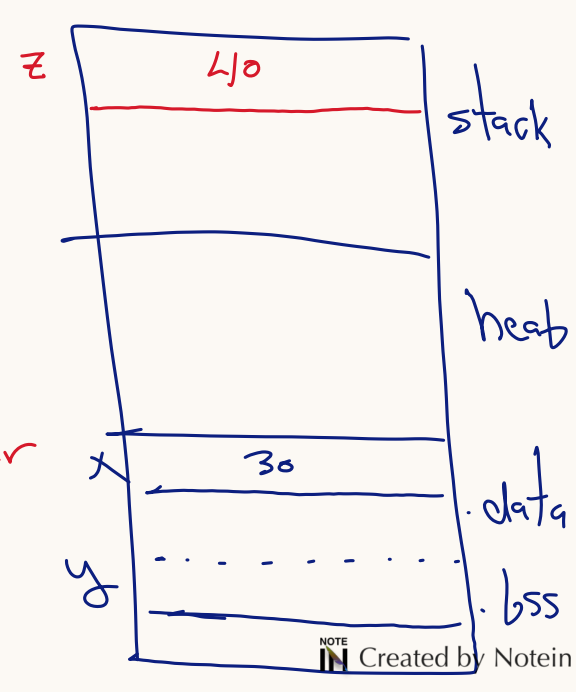
↳ var → data → Ram

char x = 30; → Definition

char y; → Decration

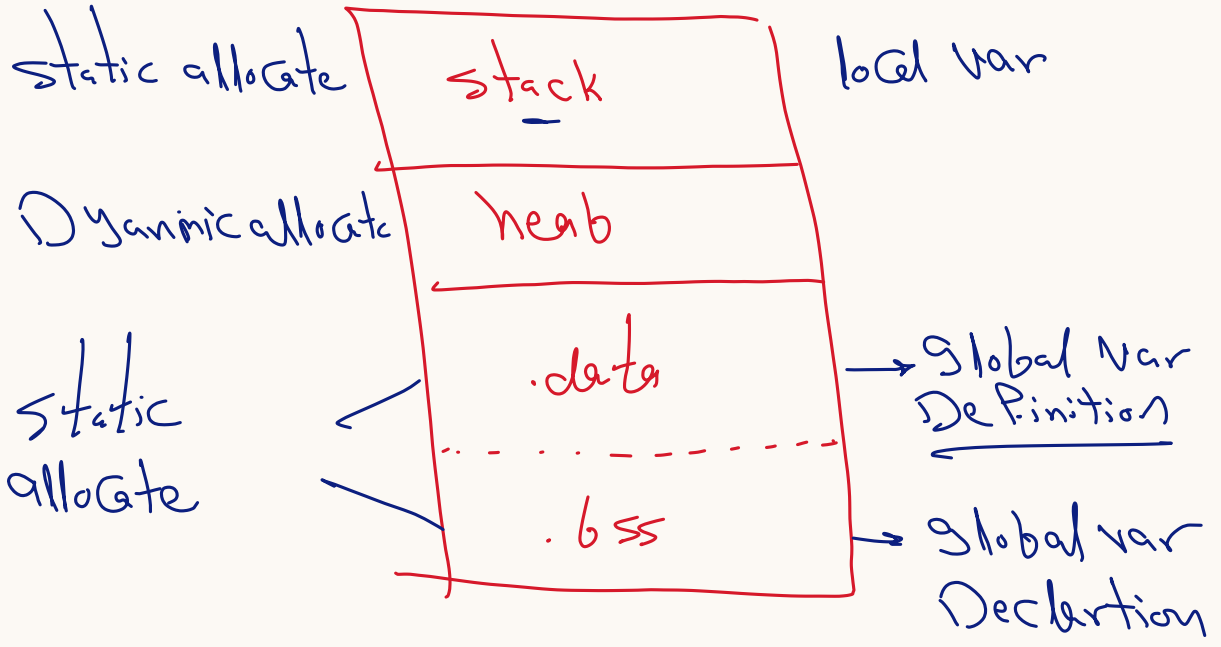
```
int main ()
{
  char z = 40;

return 0;
}
```

local var



* local var store in stack
* global var with intial value store in [.data]
* global var without intial value store in [.bss]
* note global var Decleration not store garbage but store zero by default

---



---

scope type
- Function scope | Block scope
- Program scope
- File scope

---

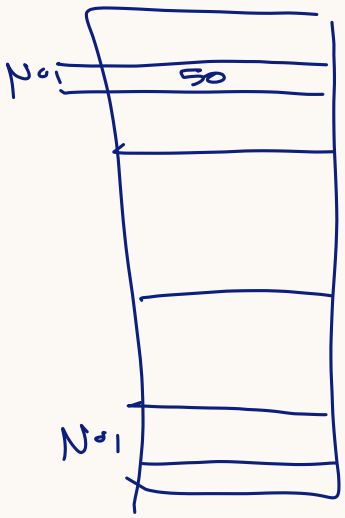
lifetime : time for var before clean.
- Function time → local var create inside Function will be Removed when Function Finished.
- Block time → local var create inside any Block will be Removed when Block Finished

↳ Program time → Global var create outside curl Block will be clear when Program finish

---

```
#include <stdio.h>
int No1 ;
```

↳ .bss



* You can't create Two local var the name inside same Block

* You can't create Two Global var the same name

```
int main()
{
  Char c = 30;
  Char c ;
```

→ error

```
   int No1;
```

→ stack

```
  Print (No1);
```

→ for example

```
}
```

Modifiers

↳ key word → effect on var & function

↳ signed ↳ size ↳ storge

↳ scope ↳ lifetime

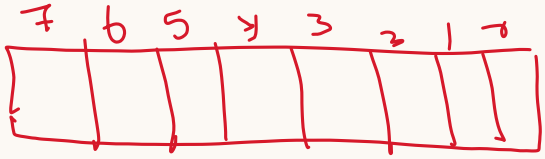
signed & unsigned

↳ ① char | int

ex char x;

signed char

store → +ve & -ve



↳ min value → 0000 0000

0

↳ max value → 1111 1111

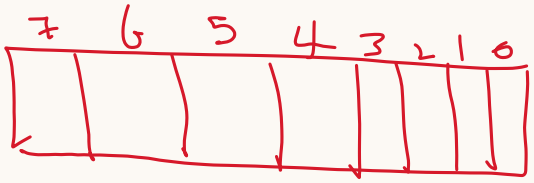
-1

signedchar

↳ -128 : 127

unsigned char

store +ve



↳ min value → 0000 0000

0

↳ max value → 1111 1111

255

unsigned

↳ 0 : 255

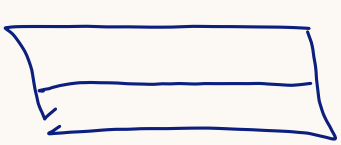
---

size Modifier (short | long | long long)

↳ short

↳ int

short int z = 50;



↳ long

↳ int | double

↳ long int z = 50

↳ long long int → 8 Byte
↳ int only

* operator → sizeof (long double)
↳ Return No of Byte

storage Modifiers → static | extern
register | volatile

*static → Global & Function & local var

static Global var
↳ scope : FileScope
↳ life time
↳ not effect
Program time

File1.c
static (s.v)

File2.c

static Function
↳ effect → scope → FileScope

static local var

↳ Scope → Note Effect
↳ Block Scope

↳ lifetime
↳ Program time

*Note → static local var will be store inside data segment

---

extern →

File.c

```
int X=20;
void Add()
{
int no1;
int no2;
no1+no2;
}
```

File2.c → build

```
extern int X;
extern void Add();
int main()
{
printf(X);
Add();
}
```

File3.c

```
int X;
```

---

register ✓

optimize for fav var

register char ch = 20;

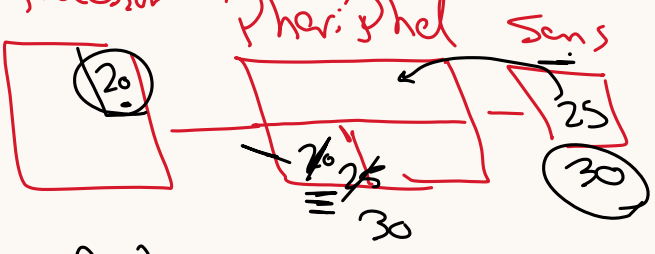
↳ Request Processor store ch inside processor

↳ Approved / Reject

↳ local var only

volatile

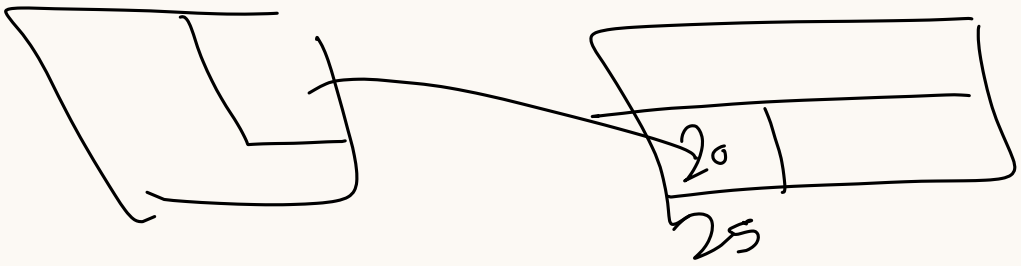
* Compiler optimize
* if find var common use will store it in processor



if temp >= 30 → Aircondi

Volatile → E.S → key word

↳ key word → stop optimization for specific Address in memory



Output & Input

↳ stdio.h

Output

```
printf(" Display ");
```

Input

① Create var

② scanf (type , Address)

↳ Display string

```
printf (" string ");
```

↳ Display Var

```
int x = 20;
printf(" x equal %d" , x);
```

① Format Specifier

↳ %d , %i → int

↳ %f → float

↳ %ld → long

↳ %lld → long long

↳ %lf → double

↳ %c → char

↳ %x → hex

```
char z = 97;
char y = 'A';
Printf(" z = %d" , z);   97
Printf(" y = %c ", y);   A
```

Printf (" z = %c " , z); → A

Scan var

↳ create var the same input type

```
char   z =0;
char   x =0;
 int   y =0;
float  a =0;
double b =0;
```

Address operator

```
scanf( " %c "  , &z ); → char
scanf( " %d ", &x ); → Number
scanf( "%d", &y ); → number
scanf( " %f ", &a ); → floating
scanf ( " %lf, &b ); → floating
```