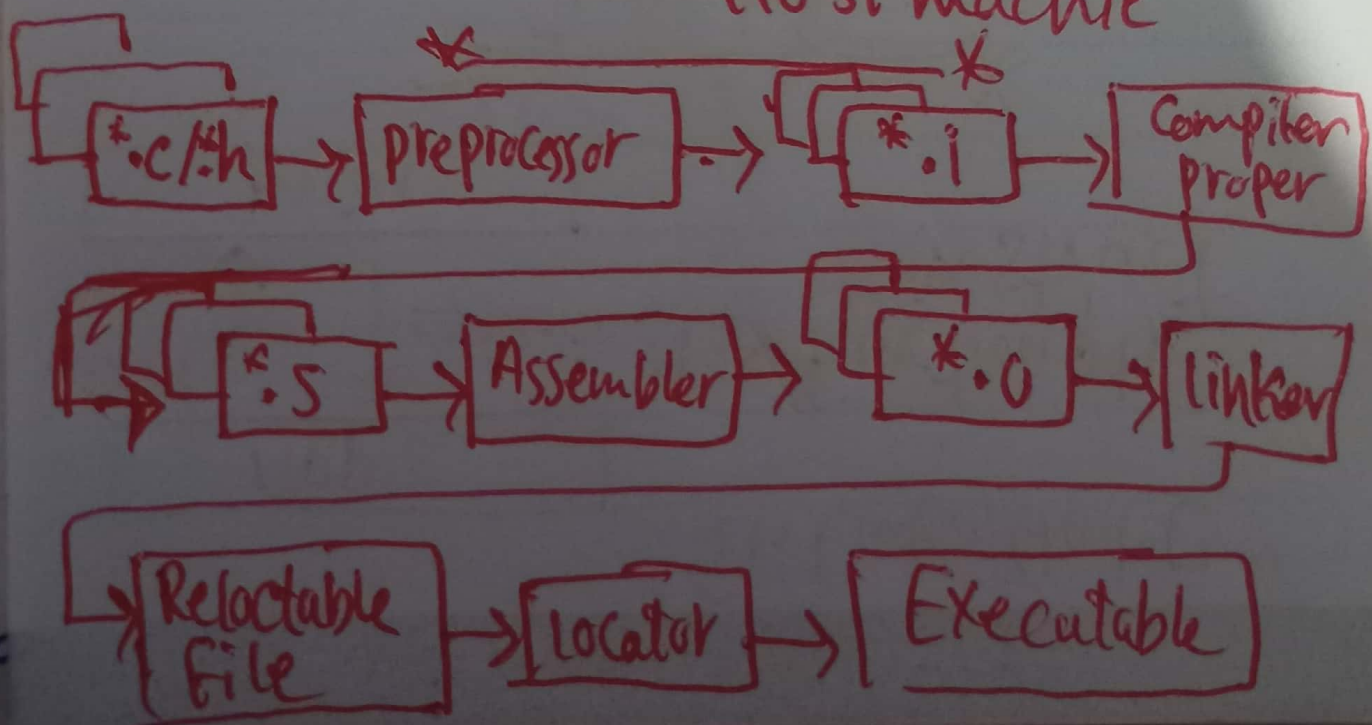
enums:

- Can be 2 enumerators with the same value.

preprocesor D

- Native Compiler:
  - generates code for the same platform on which it runs. (GCC, Turbo C)
- Cross Compiler:
  - generates [.exe] code for another platform.



tool chain:

تحويل من اللغات [.exe] بتاعه

ال Compiler وتحويل (High → low)

هتلاقيها موجودة عند ... في (bin) Code blocks

.i → intermediate file

```
cpp app.c > app.i
```

} trigger preprocessor

1- #include ⇒ copy the content of file.

- #include < > → Libs (هيدور في ال comp)
- #include " " → user-defined → .h or .c (common)
  - .c → cause multi definition Err

2- /* Remove Comments */ (هيدور في ال path ال .c)

```
gcc -S app.i
```

".S ⇒ assembley"

```
as -o app.o app.s
```

".O ⇒ object"

3- #define "Macro"

#define NAME value

حاجات متعلقاش

#define x "empty Macro" متعلوش

#define x 3;

#define x 3 //Comment

MultiLineMacro

#x

define01

TEST01 ↳ #define0TEST077

77

String Macro?

#define NAME "mostafa" عادى ✓

MultiLineMacro? Comment لو هتعمل

حطه قبل الـ '\' بتاع newline

TEST /*This is Commen*/ \

غير كده لما يأخد Copy هيأخد غلط

خلى بالك من الـ Spaces

MacroFunctions → no space

```
#define SUMMING (n1,n2) (n1+n2)
```

datatype لاحظ مفيش

↳ ده اللي هيتاخد copy

دي ميزة علو تجمع int مع f

↳ لو حطيت "3" ← مش هيطلع warning لكن undefined behavior

طب لو فانكشن عادية ← warning ومعكن error

* تفتكر ايه مشاكلها؟

- كل ما تعملها call هتاخد الـ body وتعمله copy
"حجم الكود هيكبر جدا" code duplicate

- في العمليات الرياضية "ترتيب المعاملات"
عشان تحلها ← حط الـ expression كله بين قوسين وحط
كل (name) بين قوسين
↳ symbol

when? _ time ✓ — مش هتستخدم كتير

_ "BIT_MATH.h"

Conditional Compilation

```
#if (cond) → Check macros not vars
code;
#elif ( ) / #else
#endif
```

لسه في البريبروسسر

بتنفذ الكود الكرو وتهمل neglect الفولص

- لو عايز تهمل comment حط الكود بين (#if 0 #endif)
  ↳ (zero "false")

```
#ifdef (....)
code;
#endif
```

هل الداتا دي defined ؟

File Guard: stop redefinitions

```
#ifndef _FILE_H_
#define _FILE_H_
.....
#endif
```

في طريقة تانية بس عامة شوية

Some predefined Macros:

- __FILE__ : file name
- __LINE__ : رقم السطر اللي فيه الكلام الطباعة
- __DATE__ → الداتا بيع

#error "Error!" ⇒ generate error message
أي حاجة أنت عاوزها بين (ـــــ)

#line رقم ⇒ بتغير رقم السطر "السطر اللي بعده هيبق الرقم اللي بعد الرقم الجديد"

#line رقم "اسم" ⇒ بتغير اسم الـ file + اللي فوقه

————————————
طظ

#pragma "Compiler dependent"

#pragma once ⇒ "file guard" تاني أسمه عام

#pragma GCC poison اسم ⇒ طلعلي error لو استخدم الاسم ده في الكود

#pragma GCC warning "رسالة" ⇒ طلع warning

#pragma GCC error "رسالة" ⇒ طلع error لو عمل للفايل ده included "مش منطقي"

- add mem section

  مفيدة في الـ Structure padding

- startup, exit

operators

#if defined (الاسم)

هل الاسم ده defined بالفعل باستخدام #define ؟

#endif

#if !defined (اسم تاني) "not defined"

Printf("Hi" "Hello"); تنفع أو هتتجمعلك

Stringification: → op: HiHello

#define PRINT (FN, LN) printf(#FN " " #LN)

PRINT (Ahmed, Moha) هيطبع الاتنين كـ string وفيه مسافة بينهم

ميتحطش عادي من غير كوتس

طب لو حطيت الدابل كوتس؟ هيطبعلك الاسم جوه دابل كوتس

طب لو حطيت الاسم بس!! ؟ warning يعني بطل هبدأ رجوع

Continuation operator '\':

↓

"اللي عايزه يكتبه" multiLine Macro فالكود ده؟

Concatenation: (##) #define Conc(x,y) x##y → xy

int x = Conc(8,3); ⇒ x = 83

#define (macro) إضافات

object like macro

Function like macro

(4 Syntax)

. #define identifier replacement list
#define iden(params) replacement list
#define iden(---) replacement ___
#define iden(params,...) replac ___

since Cgg

[1] #define (x) 10

var مثل

→ var نفس قواعد

good practice ⇒ CAPITAL

what if ?

#define x 10
int y=x;
int x=50;

error ⇐

```
#define z 10
#define x z
int y = x;
```

→ y = 10

طب لو بدلت أول سطرين؟
- شغال برده.

```
#define x 10
#define x 20
```

} warning ⚠

To avoid the warning:

```
#define x 10
int y = x;
#undef x
#define x 20
int z = x
```

#define	#enum
text replacement @ preprocessing	text replacement @ Compiling
Float, sentence	Const integers
no memory space	space = int size
(X)	inside switch
اسم واحد قيمة واحدة	Same value ⇒ diff names

typedef	#define
Compiler	preproccesor
only types	types, values
;	X

(...) → anything else

var ← store ↓

↳(__VA_ARGS__)

Ex:

#define fun(a,...) printf(__VA_ARGS__,a)

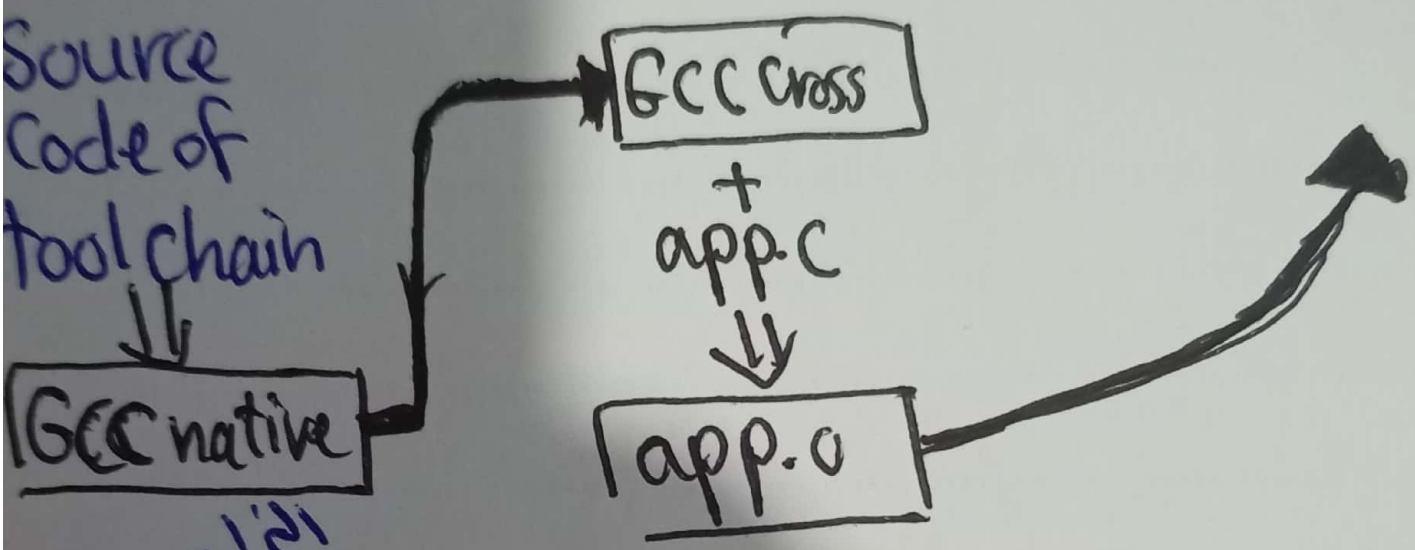
Fun(X,"X=%d"); ≡ printf("X=%d", X)

Systems

Build

Host

Target



Cmd:

gcc main.c -I /users... ⇒ include header (path)

gcc -wall main.c ⇒ enable warning

gcc -werr main.c ⇒ —— errors

gcc --help

libraries

Static

ميشوفش
غير ليه؟

خصوصية حماية

هيتعامل مع .h بس

Dynamic

يشوف

الـ .c

((How?

object

هتديله (.o)/(.a) هيدخل على اللينكر على طول

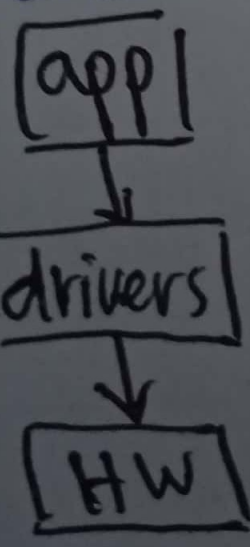
وهو عارف هيستخدم الفانكشن ازاي من .h " prototypes "

ARM Cross TC

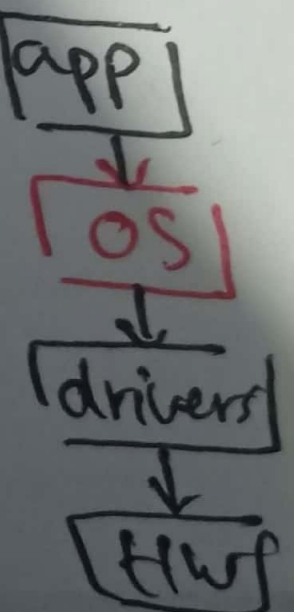
- GNU GCC (Free/openSource)
  - IDE
    - STM32
  - without
    - download
- arm cc (Commercial)

ARM TC

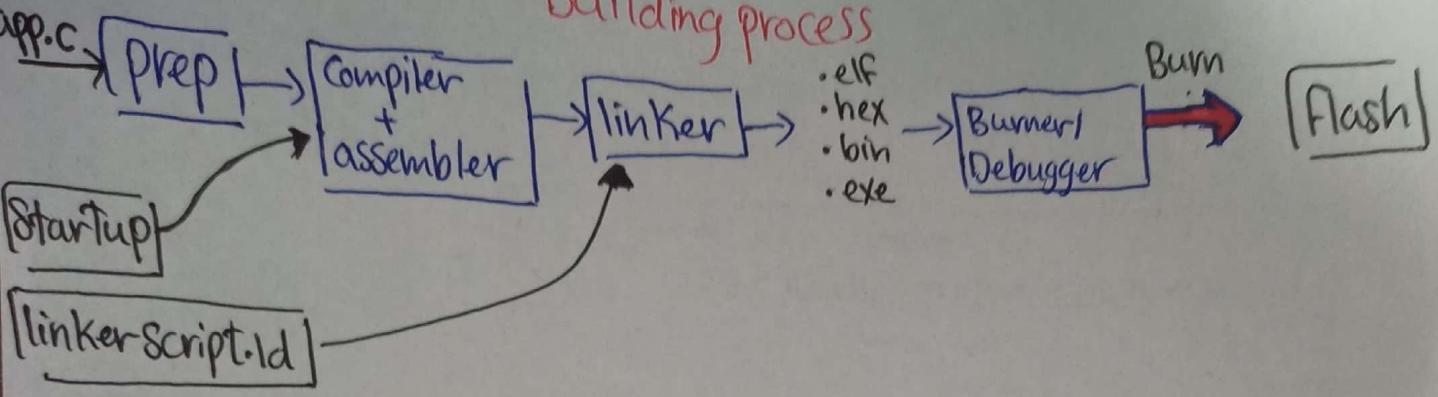
- non_abi_gcc
  - Baremetal app



- linux_gnu_api
  - OS app



Building process



1- preprocessor

app.c → prep → app.i (app.p)

Text replacement "mentioned earlier"

2- Compiler

app.i → code generation stage [Compiler] → app.s → based on the arch

3- Assembler

app.s → Assembler → app.o "Relocatable files" ⇒ Virtual Adds

app.o ← machine Code

4- Linker

libraries
app.o
linker Script
→ linker → app.exe + debug file ⇒ Bin-util: obj copy → app.bin

Virtual ⇒ physical Adds

Code Generation stage (Compiler)

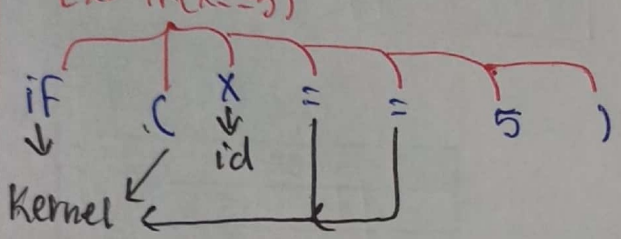
- Front-end stage
- Middle Stage
- Back-end Stage

[1] Front-end Stage: - Source Code parsing
- Check syntax errors By Tokenization

↳ Tokens { Keywords, Identifiers, operators }

OP → parse tree —Symantic table→ IR program

Ex: if(x==5)

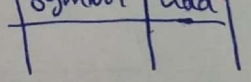


[2] Middle Stage: { Semantic Analysis {Check logical structure "datatypes"}
optimization why → {Code size, time, mem size}

multi-level process { ① Dead code remove "after return"
② Func ⇒ inline Func
③ Register allocation "GP registers"
④ loop unrolling

[3] Back-end Stage: { Code generation "Convert to assembley"
Mem allocation ".data, .bss, stack, ---"

* ——— * → virtual



⇐ linker يحل

Notes:

1- Compiler ⇒ Symbol table { var names, Func names

Import info to file = this "extern"

Export info From files = this - global, static vars - Func names

2- Compiler ⇒ Debug info ⇒ debugger يحل

3- Compiler parse only one C file at a time.

* ——— *

Linker: { Symbol Resolving ⇒ "where the symbol exists?"
Section Location: virtual ⇒ physical add using linker script

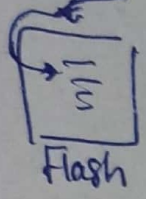
using locator

(.) dot operator

Booting Sequence

Power on/reset

① ⇒ entry point ⇓ From datasheet



(.bin) $\overset{\text{burn}}{\Longrightarrow}$ MC

- ① Startup code (.S) ✓✓ (.C) ⇒ if stack is initialized before
- ② Code → (.text)
- ③ global, static (Init) $\neq 0$ → (.data)
- ④ " " (unint) $= 0$ → (.bss)

→ For the boot loader

Ep → Boot loader: "SW without OS" → 'startup+main'

or

Your baremetal SW → "startup + app" → main

1- Baremetal SW: PC → entry point

↳ reset section of startup code

ROM mode

Basic Startup code tasks: ① Init_stack ② Copy (.data) from Flash to ram

③ (.bss) to ram "directly → never exists at Flash"

④ Jump to main

2- Boot loader: entry point ⇒ boot loader

Startup Code ⇓ نفس وظائف السابق

main

① Init_ modules

② load "Flash ⇒ OS/SW"

③ Jump ⇒ startup Code (load → RAM) → "SW"

Jump → "main"

RAM mode

ROM mode

ROM "Flash"

entry point

.exe code

Const data

RAM

Stack

Changing data

very simple
small mem
Fixed Code add
relative small Code

RAM mode

RAM

.exe Code

Stack

Changing data

ROM

entry point

Complex
relocatable Code
large Code
Fast "RAM > ROM"