# Pointers

↳ variable contain the Address of var

↳ Represent Address Type

## declaration

char *pointer = &x

1- must contain *
2- & Address operator used to get address of a variable

3- To access value stored in specific Address ⟹ use *

*pointer =10 ⟹ x =10

|  |  |
|---|---|
| 0X00 | X |
| 0X01 | Pointer |
| 0X02 |  |
|  |  |

## Type casting

طريقة بتخليني أقدر أحول من dataType
لـ dataType تاني

int ⇆ char ⇆ double ⇆ float

ex

```
float X = 12.3
int y = X        ⟹   y = 12
```

← ماينفعش تخزن قيمة float داخل int فعلشان كدا حصل Type casting

← الـcompiler بيظبطها ← int y = (int) X

ex

```
char X = 53
print X      ⟹   5 الكاراكتر اللي الـASCII بتاعها 53
int y = X
print y      ⟹   53        y = (int) X
```

## Address casting

PortA → Address 0X05

* PORTA = 0Xff ⟹ 0X05 = 0Xff

→ compiler error

لازم اعمل casting لـ 0X05 ع انه القيمة الموجودة داخل عنوان في الـ memory *((char*)0X05) = 0Xff

0X05

→ أصبح عبارة عن عنوان في الميموري
يخزن جواة char قيمة 0Xff

→ مكان في الميموري حجمه 1byte

((int*)0X20)

→ مكان في الميموري حجمه 2byte

*((int*)0X20) = 5

|  |  |
|---|---|
| 5 | 0X20 |
| 00 | 0X21 |
|  |  |

## Volatile keyword

```
main()
{
  int  X=100, y=10;
   while (X<200)
         y ++
 }
```

الـcompiler في بعض
الأحيان بيعمل optimiza ⇐
للكود ← بيقلل الأماكن
اللي بيأخدها في الـmemory
و يزود الكفاءة

← في البرنامج دا مثلاً الـ X قيمة ثابتة مش بتتغير
في البرنامج ← الـcondition مش ليها لازمة

فحينه أنه ممكن أكون مغير قيمة X (by H.W)

الأقوى

كدة الكومبيلر بعمله بيشوف التغيير بالنسبة لـ S.W كـ

← علشان كدا احنا عاوزين نفهمه انه القيمة هتتغير by H.W فابنستخدم كلمة Volatile

Address casting ⇒ *((volatile char*) 0x05)

← عبارة عن Address لـ char وارد أغير القيمة المخزنة فيه by H.W

# Defined Type

مشكلة ⇐ بتظهر لما اجي اكتب كود على 8bit mc وعايز أنقله مثلاً على 32bit mc

* ودايماً بنحاول نحافظ على الـmemory يعني مثلاً لو في حاجة محتاجة 2bytes ← ما احجزلها بس 2bytes مش أكثر

* الـcompilers بيختلف تعريفها لـ int
char
double
مما تتغير في وظيفتهم لكن مختلفين في المساحة بتاعة كل نوع

int → 2byte / 4Byte / 8byte

الحلو ← بعمل dataTypes جديدة انا اللي بعرفها

* Typedef unsigned char u8

uint8_t

* Typedef signed char sint8_t

number of bits

Type defined

Compiler 8 bit:

→ Typedef unsigned char unit 8_t
→ Typedef unsigned int unit16_t

char ⇒ 1 byte
int ⇒ 2byte

compiler 32 bit

→ Typedef unsigned char unit16_t
→ Typedef unsigned int unit32_t
→ Typedef // short char unit8_t

char ⇒ 2byte
int ⇒ 4 byte

# C Typedef

→ keyword used in C to define a new Type

```
Typedef unsigned char B
```

* use _t in the end of your defined Type to make the user know it's a ~~de~~ defined type

## why we use Typedef ??

→ code Portability ⇒ 8 bit / 32 bit ← على الcompiler

→ code Readability

```
int (*arr[4])( )={test1, test2, 3,4}

int (*(*pf()) ([4]) ( )
```

Applications → with Pointers

```
typedef int * intPtr
. intPtr ptr;
```

→ with structure

→ with function Pointer

```
typedef int (*apfArith[3])(int,int)
apfArith x = { A, b, }
```

→ with Arrays

→ with structure pointer

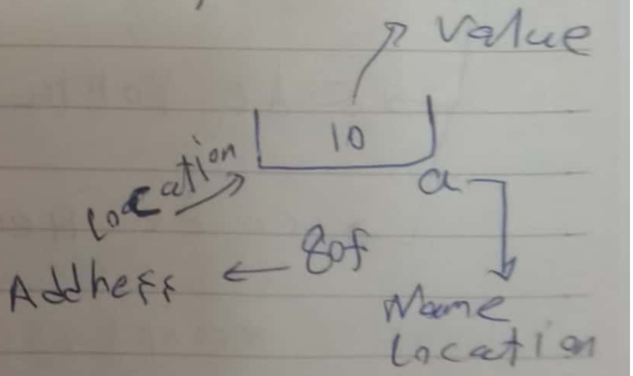
— // —

## Pointers

→ Address of memory location meaning ??

↳ each variable you define ~~has~~ allocates Location in memory

int a=10;

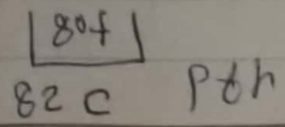


* We can access Value 10 by using

↳ variable Name

↳ Address 80f

variable that used to hold this Address ⟹ Pointer



# Pointer to Struct

```
Typedef struct Point
{
      int x; int y;
}*P;
   (*P).x   ⟺   P->x
```

## * function Pointers #

↳ function isn't a variable but it's possible to define Pointers to function

→ pass function كمان أستخدمه في إني أعمل as an argument to other functions

Array of function pointers في دي الـfunction وكمان أعمل

```
⟹ double(*pf) (double, in);
```

```
#    int (*Pf) (char)   → Pf is a Pointer
     int *f (char)
         → f is a function ~~take~~ char
              return Pointer to in
```

# Pointer to Pointer

↳ int x = 10    int *ptr = &x
int **ptr2 = &ptr

const int * ptr
↳ ptr is a pointer to constan integer

int * const ptr ⇒ ptr is a constant pointer ~~integer~~ to integer

C Arrays ⇒ variable store multiple variables

## passing Array to function

```
. void x (int * param)
. void x (int param[10])
. void x (int param[])
```

result = x (age)
↙ name of array only

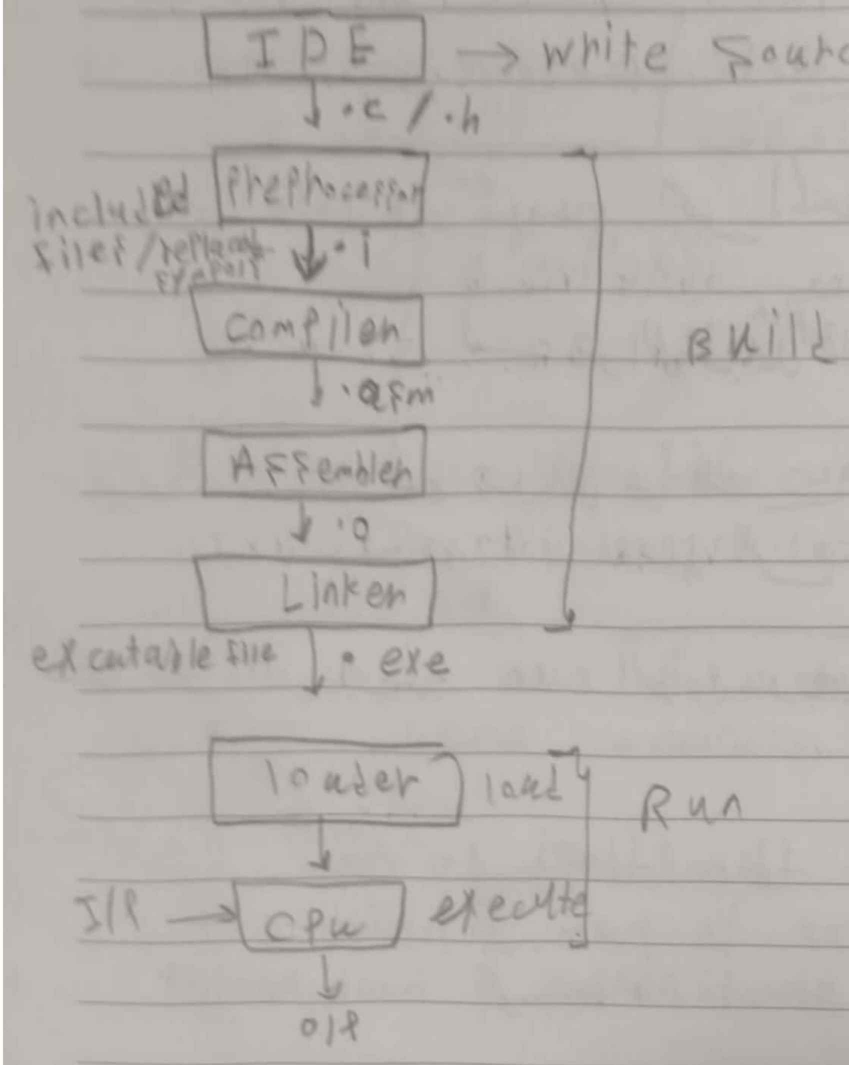
How to copy array to another array

↳ using struct

```
struct mystruct { int num[]; };
struct mystruct struct1 = {{1,2,3}};
  //        //       struct2 = struct1
```

Date: Subject:

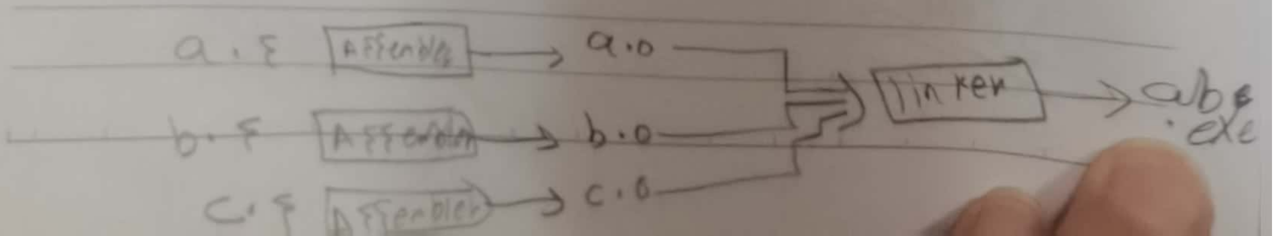
# Camilation Process & linker Schl file



⟹ Linker → while writing a multi-file pr

each file is assembled individually into object files

↳ linker combines these object files to form final executable file.



program

Date: variables Subject:

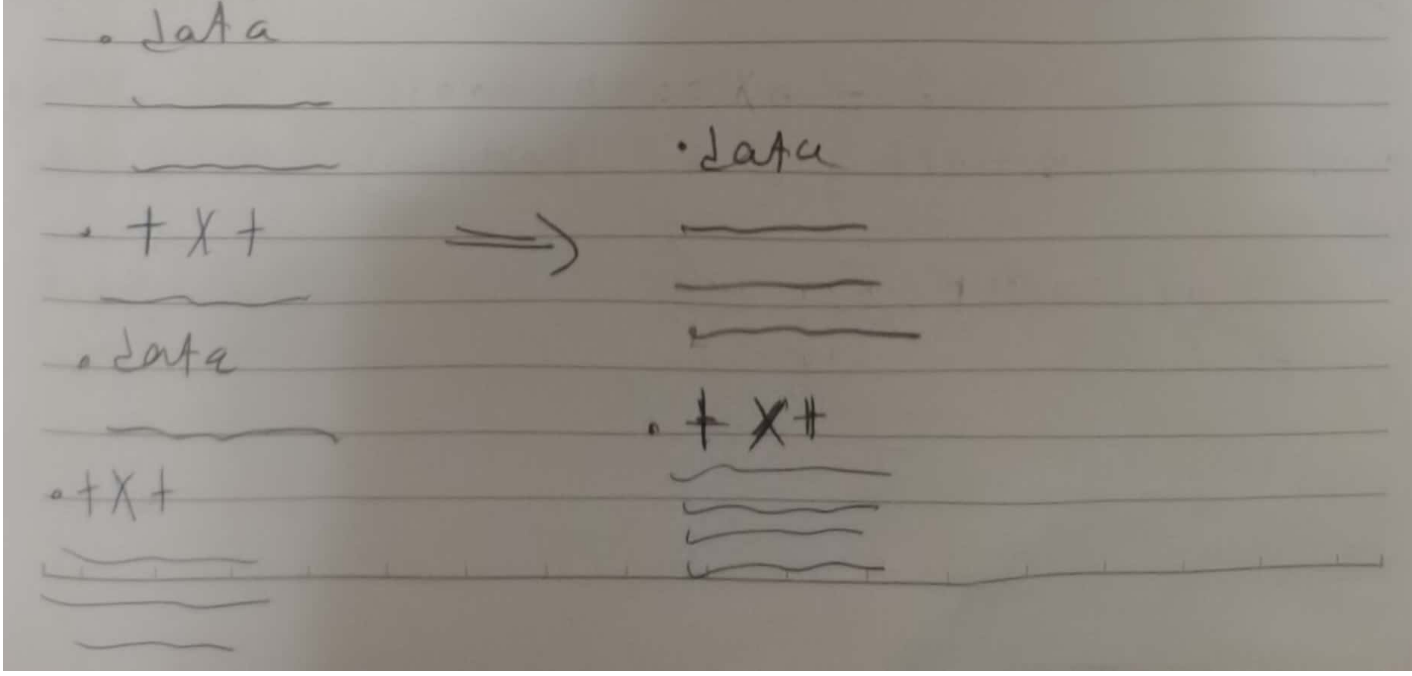
* Code & data have different run-time require...

ex
- code can be placed in read only memory
  لأن ده بحتاجه دايماً طول ما انا في البرنامج وكمان محتاج انه البرنامج
  وهو شغال ما يعدلش فيه الكود ← [memory ثابتة]
- data → read-write memory Ram
  لأنه كل شوية بحتاج أعدل فيها

⟸ علشان السبب دا بلزم الdata و الcode يكونوا منفصلين ⟸
so program are divided into sections

code ⟵ .txt
initialized global var ⟵ .data
uninitialized global var ⟵ .bss
global const var ⟵ .rodata

* .txt, .data — Assembler directives بيكون موجود عند ✗
بستخدمهم علشان اقدر احدد الجزء دا من الكود هيخزن
في اني section



1 كده هنا بقى دور الـ linker script file

- Section merging & placement is done by linker
- Programmer can control how the sections are merged, and at what locations they are placed in memory through linker script file

→ How to write linker script file

(الخطوات اللي فوق أكيد
هيعمله الكود بتاعي

```
SECTIONS {
        . = 0x00000000 ;
    .text : {
              abc.o (.text) ;
              def.o (.text) ;
    }
}
```

Or

```
SECTIONS {
        . = 0x00000000 ;
    .text : { *(.text) ; }
```

هيجمع كل الـ .text

و يحطهم ورا بعض
all text

Date: Subject:

* if program contains .data & .txt

```
SECTIONS {
        . = 0X00000000;
        .text : { *(.text); }
        . = 0X00004000;
        .data : { *(.data); }
        }
```

كتابة ~~أحط~~ .text في Rom ، .data في Ram فا وأنا بعمل

.data هروح أكون بعمل access لـ Ram address ⟹ م

مشكلة └→ volatile memory data lose

* code must be loaded into non volatile memory to save it [flash memory (rom)]. but variables should be stored in [Ram] → so they can be easily modified.
└→ it's a volatile memory so it's not possible to save data in Ram.

so ⟹ All code & data should be stored in flash before power ~~on~~ up

* الكلام ← ~~اكتب~~ كود لما أكتب البرنامج أنزل .text و .data
في Rom و أكتب كود startup code ← ويتنفذ قبل ما أنفذ الكود
بتاعي
وظيفته ← يأخذ ~~يعمل~~ copy من الـ data الموجودة
في Rom ويحطها في Ram وبعد كدا البرنامج بتاعي يشتغل على الـ data
الي في Ram ويعمل update في Ram

Date: Subject:

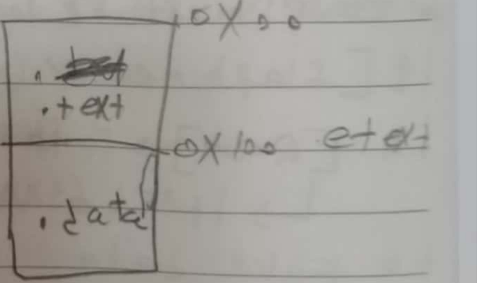
علشان كدا ← كل var موجود في data. هيكون ليه 2 Address
→ Load Address ← in flash (Ram)
→ run time Address → in Ram

* كدا هـ أعدل في ال linker script بتاعه.

```
Sections {
        0X00;
        .text : { *(.text); }
        ~~estte~~ etext = . ;
        data : AT (etxte) { *(.data); }
}
```

← بأخد عنوان آخر مكان في rom المستخدم أكتب فيه زي الـ var

AT → بتعامل مع load Address



* بعدين بكتب كود ال startup ← ولازم أحدد ال address للـ data في Ram و flash وال size بتاعها

flash_sdata ram_sdata data_size
start
→ linker script file هو اللي بيحدد العناوين دي

```
Sections {
.=0X0000;
 .text : { *(.text); }
flash_sdata = . ; (1)
. = 0XA0000000;
ram_sdata = . ; (2)
.data : AT(flash_sdata) { *(data); }
ram_edata = ; (3)
data_size = ram_edata - ram_sdata.
```

startup code is done موجود في الآخر

Date: Subject:

# Start up code

→ it's not possible to directly execute C code مخزنة محتاج في الأول اخذ الداتا بتاعتي انقلها
من Rom لل Ram

1- Stack → can handle with local variables
علشان ال stack دا مش محتاج اعمله copy من flash لل Ram ←
[أي حاجة داخل main] ←

2- Global variables → Initialized
→ un Initialized

3- Read only data

Startup code
- → code to copy .data flash to ram
- → code to zero out .bss
- → branch main
- → code to setup stack pointer