

Lec 7

Casting : converting one datatype into another

هو عملية تحويل من Datatype لـ Datatype ثانية

ولها اسم تاني هو type-conversion

* عليه التحويل ممكن تتم بشكلين :

① Implicit type casting

• تتم من خلال ال compiler

automatically

• ممكن ينجح او يفشل

② Explicit type casting

• تتم من خلال ال user

(type-name) expression

• فحاله فشل ال compiler فابوجه

يقول ال compiler اصل casting

حتى لو هيحصل Data loss

to avoid the loss of information → Following the rules

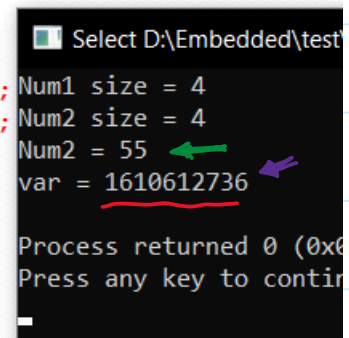
Rules :

- ① All integer types to be converted to float
- ② All float types to be converted to double
- ③ All character types to be converted to integer

```
#include <stdio.h>

unsigned int Num1 = 55 ;
unsigned long Num2 = 66 ;
float var = 3.14 ;
int main()
{
    printf("Num1 size = %i \n", sizeof(Num1));
    printf("Num2 size = %i \n", sizeof(Num2));

    (1) → Num2 = Num1;
    (2) → Num1 = var ;
    printf("Num2 = %i \n", Num2);
    printf("var = %i \n", var);
    (3) →
```



هنا متغيرات من انواع مختلفة

55
 في رقم [1] بخزن قيمة من متغير `int` في متغير `long int` هتلاقى اى `compiler` عملها `casting` عادى وطبع
 في رقم [2] بخزن قيمة `float` في متغير `int` فطبع هتلاقى المتخزن اى `int` وبيرد اى `float` هيرجع
 فضا اى `compiler` فشل يعتبر انه يحافظ على القيمة لأنها بقت 3 مش 3.14 (مقدرش يعمل `casting`)
 في رقم [3] اى `var` دا من نوع `float` أنا عاوز اطلع اى `integer` هتلاقى اى `output` اعزير تمامًا
 مثاها مقدرش يعمل `casting`

```
printf("Num1 = %f", Num1); Num1 = 0.000000
```

هنا قيمة `integer` حاولت اطلعها اى `float` كان اى `output` بالشكل دا
 يعنى مقدرش يعمل `casting`

هنا اى `compiler` فعلا ساعات بيقرر يعمل `casting`
 و ساعتها بيكون اسمها `Implicit casting`

Rules of casting

Wednesday, October 18, 2023 2:00 PM

* التحويل يتم من تحت ل فوق (يعني من الصغير إلى الكبير)

* لو حصل العكس بيحصل (data loss)

لأن كما بخرن data حجمها كبير في data type حجمها صغير

لو عندي Expression فيها أكثر من كذا operand المعادلة كلها بتتحول

لا operand إلى حجمه كبير كله بيتحول لأكبر نوع

والناتج بيكون من النوع دا برضو

Note:

* لو هشتت من فوق لتحت يعني من higher data type إلى lower data type

هنفقد كذا bits

• Moving From double to float causes rounding of digits.

• Downgrading From float to int causes truncation of the fractional part.

```
#include <stdio.h>
#include <stdlib.h>

unsigned int NumberOne = 7;
unsigned int NumberTwo = 2;
unsigned int Result = 0;

int main()
{
    Result = NumberOne / NumberTwo;
    printf("Result = %i \n", Result);
    printf("Result = %f \n", Result);

    return 0;
}
```

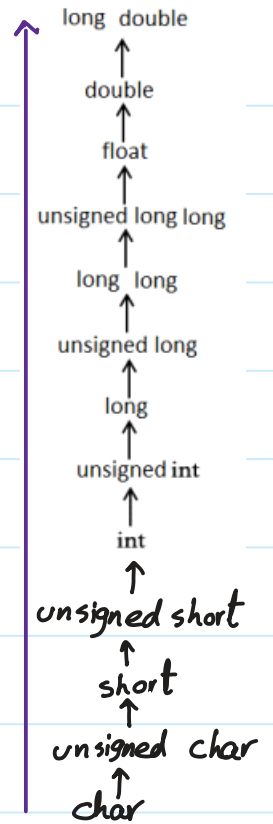
```
Result = 3
Result = 0.000000

Process returned 0 (0)
Press any key to cont
```

في المعادلة دي ① من نوع unsigned int وكذلك ② : الناتج هيكون من نوع unsigned int

فهتلاص الناتج بدل ما المنوع 3,5 بقا 3 عشان integer ولما اطبع كد int ← ظهر 3

و لكن لما برينا نطبعه كد float هقدرش يحوله وطبع الشكل دا 0,00000



```
#include <stdio.h>
```

```
unsigned int Num1 = 7 ;
```

```
unsigned int Num2 = 2 ;
```

```
float Result = 0 ;
```

```
int main()
```

```
{
```

```
Result = Num1 / Num2 ;
```

```
printf("Result = %i \n", Result);
```

```
printf("Result = %f \n", Result);
```

D:\Embedded\test\test\here.c

Result = 0

Result = 3.000000

Process returned 0 (0x0)

Press any key to continue.

في المعادلة دي المتغير `Result` من نوع `unsigned int` وكذلك `Num1` و `Num2` : كذا الناتج من نوع `unsigned int`

يعني $(7 / 2 = 3)$ وليس 3,5 كما هو المتوقع وخت الناتج باحطيه في `Float`
 $\downarrow \downarrow \uparrow$
`unsigned int`

اما جيت المبع ك `int` مقدرش يحول من `Float` إلى `int` (نوع المتغير المحفوظ فيه)
 واما طبخته ك `Float` طبع عادي 3 ولكن ك `Float` فكتب 3,000000

عدال `bits` بعد العلامة ل `Float` ← 6

هنا المشكله ان المفروض الناتج كسر ولكنه طلع صحيح بالرغم ان ممكن الحفظ كان `Float`
 وضاع الكسر بالنتيجة ان ال `operand` هزار `Expression` كلهم من نوع `int` فالناتج `int`

الحل في المشكله دي ان انا ابي اعمل ال `Casting`

يعني `Explicit casting`

الـ **Explicit casting** هو إني اروح للمتغير واحوله لنوع الـ **data type** اللي عاوزه يبقى عليها

```
#include <stdio.h>

unsigned int Num1 = 7 ;
unsigned int Num2 = 2 ;
float Result = 0 ;
int main()
{
    Result = (float)Num1 / (float)Num2 ;
    printf("Result = %i \n", Result);
    printf("Result = %f \n", Result);
}
```

```
Select D:\Embedded\test\test\h
Result = 0
Result = 3.500000 ←
Process returned 0 (0x0)
Press any key to continue.
```

Explicit casting

هنا انا عاوز الناتج يكون **float** بالرغم من ان الـ **operands** من نوع **int**

فجيت قدام المتغير وكسبت كذا () نوع الـ **data type** اللي عاوزة يكون عليه فالقته دي بس

هو متحولش لـ **float** لا هو زي ما هو **int** ولكن في العملية دي هيعمل كـ **float** فمش هيفقد الكسر

في حالة الـ **compiler** فشلت انه يعمل **Implicit casting** هش بيديك **Error** هو بـ **Run** عادي

ولكنه بيديك **Warning**

Ex:

```
Line Message
=== Build: Debug in Test2 (compiler: GNU GCC Compiler) ===
In function 'main':
12 warning: format '%f' expects argument of type 'double', but argument 3 has type 'unsigned int' [-Wformat=]
12 warning: format '%f' expects argument of type 'double', but argument 3 has type 'unsigned int' [-Wformat=]
15 warning: format '%i' expects argument of type 'int', but argument 2 has type 'double' [-Wformat=]
15 warning: format '%i' expects argument of type 'int', but argument 2 has type 'double' [-Wformat=]
=== Build finished: 0 error(s), 4 warning(s) (0 minute(s), 0 second(s)) ===
```

يفضل اني مديش فزيمه لـ **compiler** انه يعمل **Casting**

واني اشوف الـ **Expression** ولولقيت حاجه من المتغيرات غير بعض وهيعمل **casting**

انا بنفسى اعمل **Explicit casting** لنوع الـ **data type** اللي عاوزه

```
#include <stdio.h>
```

```
unsigned int Num1 = 7 ;
```

```
unsigned int Num2 = 2 ;
```

```
unsigned int Result = 0 ;
```

```
float Resultf = 0, var1 = 24.5, var2 = 7.2 ;
```

```
int main()
```

```
{
```

```
    3      1      2  
    Result = Num1 / Num2 ;
```

```
    printf("Result = %i - %f \n", Result , (float)Result);
```

```
    4  
    Resultf = (float)Num1 / (float) Num2;
```

```
    printf("Resultf = %i - %f \n", (unsigned int)Resultf , Resultf);
```

```
    5  
    Result = (unsigned int)var1 / (unsigned int) var2 ;
```

```
    printf("Result = %i \n", Result);
```

D:\Embedded\test\test\here.e

Result = 3 - 3.000000 ✓

Resultf = 3 - 3.500000 ✓

Result = 3

Process returned 0 (0x0)

Press any key to continue.

2¹ من نوع unsigned int :: الناتج سيكون unsigned int وحفظت الناتج في متغير unsigned int ③

جيت الطبع المتغير أول مرة ك int فطبع على ولما جيت ابعده ك Float وهو في الاساس int

Explicit
casting

كدا هيحصل casting فمش هسي فرصه ل compiler انه يغلط فعملت

ف ④ انا عارف ان الناتج كس وعاوز احافظ على الكسر فعملت Explicit casting ل Float

فكدا بقا انا operant في ال Expression كلها Float :: الناتج هيكون Float وانا مكان خزنه في Float

بعد كدا جيت الطبع انا Float را ك Int فعملت Explicit casting ل int

ف ⑤ انا operants كلها Float فـ الناتج Float ولكن انا هخزنه في integer

فكدا هيحصل casting فعملت Explicit casting ليه ل unsigned int

فالناتج يكون من نفس النوع و خزنه في نفس النوع

فكدا انا اناقصدت value ولكن بهزاي وانا عارف مش من compiler

Memory layout Quick introduction

• أي project بكتبه بيتكون من data و code يكونوا جوا ال memory

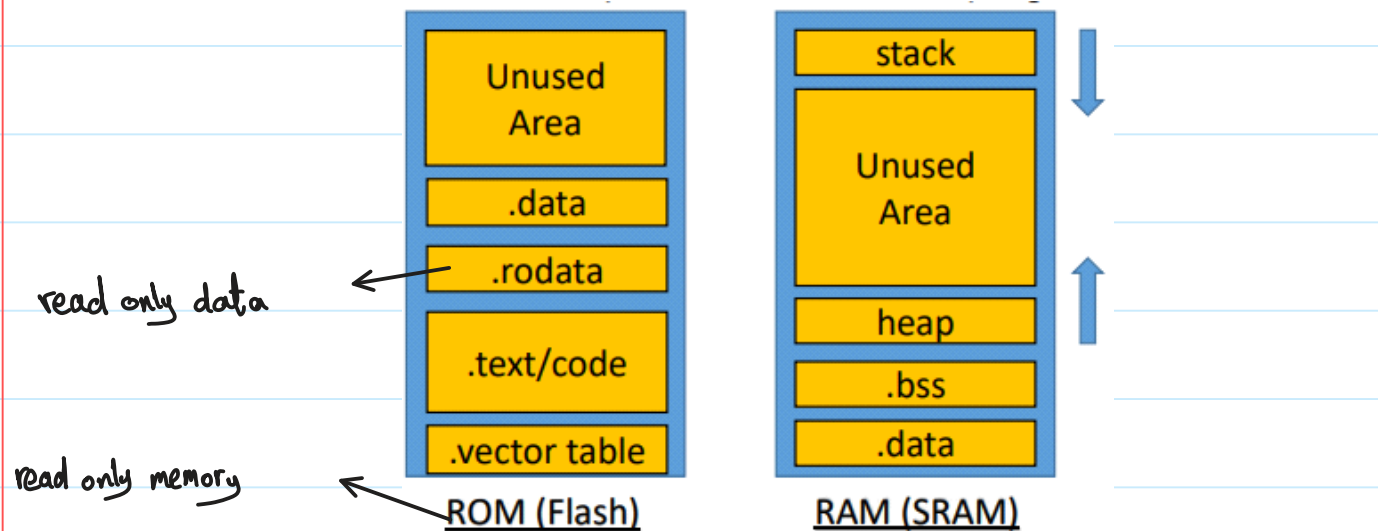
• أي micro controller على الدقل بيتكون من نوعين من ال memory

• Flash

بيتخوى على ال code اللى processor هيعمله

• RAM

بيتخوى على ال data اللى هتعمل عليها تنفيذ ال operation



• .text/code ← بيتخزن فقط فيه ال code وال Functions "زي ال main() و ال Function بعلمها"

• .rodata ← فيها ال data اللى مبيتغيرش ال const

• heap section موجودة جال RAM

• بتستخدمها من dynamic memory allocation "يعني لما أجبز مساحة اثناء ال Run time"

• كل section في ال memory بيتخزن في حاجة سواء ROM أو RAM

• عندى انواع كتيرة من ال data ، كل نوع له مكان مخصص لتخزينه في ال memory

```
unsigned int Number1 = 55; /* Global Variable Initialized */
unsigned int Number2; /* Global Variable Not Initialized */
unsigned int Number3 = 0; /* Global Variable Initialized To 0 */
const unsigned int Number4 = 55; /* Constant Global Variable Initialized To 55 */
static const unsigned int Number10 = 99; /* Static Constant Global Variable Initialized To 99 */

int main()
{
    unsigned int Number1 = 55; /* Local Variable Initialized */
    unsigned int Number2; /* Local Variable Not Initialized */
    unsigned int Number3 = 0; /* Local Variable Initialized To 0 */
    const unsigned int Number4 = 55; /* Constant Local Variable Initialized To 55 */
    static unsigned int Number5 = 55; /* Static Local Variable Initialized */
    static unsigned int Number6; /* Static Local Variable Not Initialized */
    static unsigned int Number7 = 0; /* Static Local Variable Initialized To 0 */
    static const unsigned int Number8 = 55; /* Static Constant Local Variable Initialized To 55 */
}
```


1 unsigned int Number1 = 55; /* Global Variable Initialized */

لو عملت ↓ 1. لو لم يتم استخدامه او عملت عليه اي عملية "بعض الـ compilers بتقبضو مش موجود وعلقتو، معانة"

2. لو تم استخدامه عملت عليه operation اي حاجة "وجينا نشوف مكانه في الـ memory"

هلاق منة نسختين * نسخة في الـ Flash ← .data مساحة 4 byte

* نسخة في الـ RAM ← .data مساحة 4 byte

Memory Regions Memory Details			
Selection: 4 B			
NumberOne			
Name	Run address (V...	Load address (LM...	Size
FLASH	0x08000000		1024 KB
.data	0x20000000	0x080004a0	4 B
NumberOne	0x20000000	0x080004a0	4 B
RAM	0x20000000		128 KB
.data	0x20000000	0x080004a0	4 B
NumberOne	0x20000000	0x080004a0	4 B

← موجود منة نسختين عشان لو حصل قطع في الكهراء الـ data متفقدش

← بتبقى موجودة في الـ ROM "ودي زائرة بتحتفظ بالـ data بعد انقطاع الكهراء"

← النسخة الموجودة في الـ ROM تظل ثابتة وفي حالة اي تغيير اي عملية على الـ variable يتم على نسخة الـ RAM

اول ما يوصل كهرا على الـ micro controller في code بـ Run على طول

اسمه start up code

← وظيفته يعمل copy لـ data اللي موجودة في الـ ROM و يخزنها في الـ data الموجودة في الـ RAM

و بيجعل Run قبل الـ main

2 unsigned int Number2; /* Global Variable Not Initialized */
unsigned int Number3 = 0; /* Global Variable Initialized To 0 */

Memory Regions Memory Details			
NumberOne			
Name	Run address (V...	Load address (LM...	Size
RAM	0x20000000		128 KB
.bss	0x20000000		40 B
NumberOne	0x20000024		4 B

التوعيين دول بيتغزنوا في الـ RAM في الـ .bss

ودي النسخة الوحيد ليهم "مش موجودين في الـ Flash"

.bss يكون موجود فيها الـ global variable

← Initialized=0 او not initialized

اللي بيحطهم
دا هنا بعض
هواد
start up
code

* يفضل لو هخليه = zero انه اسمه not initialized عشان في compiler ممكن تعتبر الميزر قيمة و نتجزة مكان زي 1

3] `const unsigned int Number4 = 55; /* Constant Global Variable Initialized To 55 */`

النوع \rightarrow value. بيتجزله مساحة في ROM في `.rodata`. "علشان كذا متي بقدر اعدل فيه"

Memory Regions			
Memory Details			
Number_1			
Name	Run address (V...	Load address (LM...	Size
FLASH	0x08000000		1024 KB
rodata	0x080004b4	0x080004b4	4 B
Number_1	0x080004b4	0x080004b4	4 B

في الطبيعي ال `compiler` مش بيظهر مكانه في
ولكن في طرق بتخليس اعراف و اجبر ال `compiler`

4] `unsigned int Number1 = 55; /* Local Variable Initialized */`
`unsigned int Number2; /* Local Variable Not Initialized */`
`unsigned int Number3 = 0; /* Local Variable Initialized To 0 */`
`const unsigned int Number4 = 55; /* Constant Local Variable Initialized To 55 */`

stack section كل الانواع دي بيتجزلها مكان في ال `stack`

كدا المفروض تكون عرفت ليه لو طبعيت `Number2` بيديك `garbage value`

"لأن متخزنه في ال `stack` ختمت يكون في `data` سابقه و متمسكتش"

و علشان كذا برضو اقدر اغير قيمة ال `local const variable` "لأنه متخزن في ال `stack`" بطريقة غير مباشرة

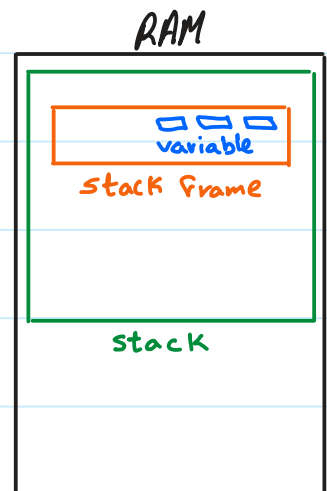
ال `local variable` اقدر اغير قيمتها بطريقة مباشرة

* لو غدي `Function` عملتها وجيت اعطها `call` في ال `main()` اللي بيحصل كالاتي:

* بمجرد ال `call` ل `Function` في ال `main` يروح على ال `RAM`

و خصيصاً جزء ال `stack` و يجهز مساحة ل `Function (stack frame)`

و يخزن كذا ال موجوده فيها ل `local variables` او `const` بيكون ليهم `Mark` ^{انهم} `constant`



* قبل ما تعمل `call` كذا دا مكنتي موجود في ال `stack` "وقت ال `call` بيتجزله مساحة"

* بمجرد ما ال `Function` تقص وظيفتها و `Return`

كل دا بيتنسخ تلقائياً من ال `stack` و ال `stack frame` بيتنسخ

* لو عملت `call` ثاني ل `Function` كل دا بيتنسخ ثاني

stronge classes

scope and lifetime of variables

میں یقیناً بشوفہ

میں محفوظی کی ال memory

* لما بن create متغیر جدید فی حاجتیں مہمیں معاد

1- Data type

2- storage classes

storage classes decides the extent (lifetime) and scope (visibility) of the variable in the program.

Local variable

* هو variable داخل [] ← scope معین
ای حد یقیناً بشوفہ تہہ داخل نفس ال []

* مقدرش اعمالہ access خارج ال []
لأنہ بیتمسح فی ال Run time

* عثمان کدا یفضل ال local variable تگون
فی بدایہ ال Function

Global variable

* اقدر اشوفہ فی ای مکان فی البرنامج
ویکون فوق ال main

* نفع يكون في أكثر من **local variable** بنفس الاسم ولكن كل واحد في **scope** مختلف

وكل واحد يأتى داخل ال **scope** بتاه فقط

* اما ال **Global variable** مينعش يتكرر بنفس الاسم هيفلع ← **(Error redefine)** لأن الكل شايفه

* برضوا ال **local** مينعش يتكرر نفس الاسم داخل نفس ال **scope** في **(Error redefine)**

* ممكن تعمل **scope** داخل **scope** **(nested scope / inner scope)** ^{اسمه} وأكثر

* لو اتا داخل **inner scope** ← اقدر اشوف اى حاجه فوقى **(outer scope)** ↑

* لما بستخدم **Variable** داخل **Function** او **scope** معين ← بيشوف اقرب **definition** ليه ويستعمله بعد ما يوصل لـ **global**

* اى **Scope** او **Function** بخرج منها ال **Variable** اللى فيها وقيمهم بتتسج من ال **memory** فن ال **Runtime**

* كذاك ال **main** مجرد ما اخرج منها (بقفل البرنامج) كل حاجه خاصه بيها فن ال **memory** بتتسج

```
#include <stdio.h>

unsigned int Number = 99 ;

int main ()
{
    unsigned int Number = 11 ;
    printf("Number = %i \n", Number); // 11
    {
        printf("Number = %i \n", Number); // 11
        unsigned int Number = 22 ;
        printf("Number = %i \n", Number); // 22
        {
            printf("Number = %i \n", Number); // 22
            unsigned int Number = 33 ;
            printf("Number = %i \n", Number); // 33
        }
        printf("Number = %i \n", Number); // 22
    }
    printf("Number = %i \n", Number); // 11

    return 0;
}
```

```
Number = 11
Number = 11
Number = 22
Number = 22
Number = 33
Number = 22
Number = 11
```

النوع الأول

stronger classes

auto (automatic)

* ال automatic

اي local variable هو auto by default

مجرد ما ار scope بتاعها ينتهي هتتمسح من ال memory بشكل automatic

auto ممكن اكتبها بولا (رئته by default بولا)

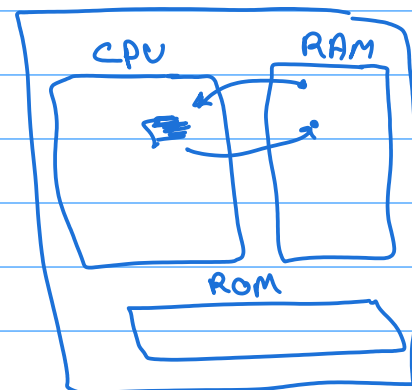
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 auto unsigned int number_3 = 44;
5
6 int main()
7 {
8     unsigned int number_1 = 44;
9     auto unsigned int number_2 = 55;
10    return 0;
11 }
```

error
متان هب
Global
وار
بيكون لا
فقط local

"Auto (Automatic)"

- The variable defined using auto storage class are called as local variables
- A local variable is in Auto storage class by default if it is not explicitly specified.
- The scope of an auto variable is limited with the particular block only
- once the control goes out of the block, the access is destroyed
- This means only the block in which the auto variable is declared can access it.

عشان اعمل اي process على متين باخدناه copy
انقله لـ CPU من register ويعمل عليه ويرجعه لـ RAM تاني
ذكر دا بياخد وقت
لو انا عاوز اخل ال variable دا موجود في ال CPU عشان اوفر وقت
بستخدم كلمة register (النوع الثاني)



Stronger classes

النوع الثاني

• register

24
25
26 → register unsigned int counter = 0;
28 for(counter = 0; counter <= 8000; counter++){
29
30 }
31
32 Ptr = &counter;
33
34 }

local فقط
مقرش اجيب
عنوانه (ملوش)

* بقتح على ال compiler انه يزن ال variable داخل ال cpu (خا العنصر الى بنفذ عليه operation)
ال compiler بيدور يزن او لا على حسب لولاق register فاض

* بعمل كذا علشان اوفر وقت Time. بتاع ال loading انه ياخذ copy منه ال RAM لا ال cpu

* تستخدم مع ال local فقط "مقرش استخدمها مع ال Global" → Error

* ال compiler بيدور على ال Register فاض يزن فيه لو ملقاش هيخزنه في ال stack

* بستخدمها غالباً مع ال For loop ← لو عشي variable بعمل عليه كل شوية زي ال counter
فبعمل register علشان اوفر الوقت وابقا اسرع من التنفيذ

* ال register داخل ال cpu ملهاش عنوان او access ومقرش اوصلها حتى لو استخدمت pointer
لأن اساساً ملوش عنوان فده جيب error

النوع الثالث

Stronger classes

extern

يستخدمها ليعاوز اعمل declare variable و declare function (من بيتجزله مساحة)

```
#include <stdio.h>
#include <stdlib.h>
```

```
extern unsigned int number_1;
```

```
int main()
```

```
{
    number_1 = 99;
    printf("number_1 = %i \n", number_1);
    return 0;
}
```

Run لو حاولت اعمل

من هيشغل

Linker error

لأنه فضل يدور على number_1

من لايه مكان عن ال memory

متجزى له مساحة

```
extern unsigned int number_1 = 9;
```

لوعملت كذا ؟

أصبح definition و ال extern ملغى لازم

متجزى له مساحة

Multi Files

بجاء ال extern واستخدمه في حالة ال

main.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
extern unsigned int Motor_Var_1;
```

```
int main()
```

```
{
    printf("Motor_Var_1 = %i \n", Motor_Var_1);
    return 0;
}
```

Motor_Var_1 = 88

motor.c

```
#include "motor.h"
```

```
unsigned int Motor_Var_1 = 88;
```

ولكن هما من يتستخدم بالشكل ده ...

يستخدمها باني جعل File.h و File.c من ال .h يعرف المتغيرات بـ extern

وجوا ال .c بعمل ليها definition واستخدمها

واجب من ال main اعمل #include "..."

للفايل

النوع الرابع

stronger classes

static

بستندها مع ال Function او ال Variable
Global local

*with Global variable:

Global variable اقدر استخدمه في اي مكان
و اي File داخل ال project
"project scope"

```
#include "motor.h"
unsigned int Motor_Var_1 = 88;
```

Global برفضو ولكن اقدر استخدمه داخل ال File بس
"File scope" اقدر استخدمه في اي مكان داخل ال File دابس
و لكن مش في باقي ال project

```
#include "motor.h"
static unsigned int Motor_Var_1 = 88;
```

*with Function:

نفس الكلام لو معرف Function داخل File اقدر استعملها على جدا داخل ال main
طبعا بعد ما اعمل include ال h. الى فيه
* لكن لو كتبت قدامها static فاقدر استخدمها فقط داخل ال File بناءها

*with local variable:

```
void Print Motor Var 1(void){
① → unsigned int Motor_Var_2 = 88;
② → static unsigned int Motor_Var = 99;
```

Stack Frame

لو خدني variable زي دا داخل Function

رقم ① هيتجزله مساحة على جدا داخل ال memory و بيتسج بجزر ما ال scope بناءه يخلص

رقم ② بيتجزله مساحة داخل ال data section. ولما ال scope بناءه يخلص هيقفل مكانه مش هيتسج

يعني ال lifetime بناءه هو نفسه بناء ال project "في ال Rom"

بيحتفظ بالقيمة بناءه between ال Function Calls يعني مش بيرجع لأول قيمة مع كل مره انادي ال Function

لا بيكمل على القيمة القديمة

Notes

* فی ال **File.h** متعلقش ای definition

سواء کن Function, variable "خلفیه ال declaration بی"
علشان لو عملت include ای ال **File.h** آکره مره میجوش redefine error

* ینفع عمل include ل **File.c** وکن مره واحد بی "آکره مره هیجول redefine error"
علشان کدا یفعل فعل include ل **File.h**

* ال **File.h** الخاص بار Module بتاعه لاسم تانی

Master Interface "header File"

* بکتب فی ال **File.h** الحاجه الی عاوز ال User یشوفها

* لو فی variable فی ال **.c** واین استفاده فی ال main "لازم عمل declaration فی ال **.h**"

* ال static بتلغی تایش ال extern

خلفی بالک ال static ← File scope

یعنی لوکان فی variable وعمله extern فی ال **.h**

وجیت فی ال **.c** عمله static هیتلغی تایش ال extern

ویش هعرف استخدام ال variable دامز ال main لائن بقا File scope

علشان اقدر استفاده بفعل فانکشن ونستدیه مره خلالها **getter Function** ترصلی قیمته

* لو عمل حاجه static یفعل تکنون فی ال **.c** سواء کن Function او variable

↑ عمل ال definition و ال declaration جوا ال **.c** ویش فی ال **.h**

لئن ال **.h** بتشوفه مره ال main ویش دای اشوف حاجه static ویش هقدر استفاده

* لوحدی global static variable واوز اعدل قیمته مره ال main بعمله **setter Function**

استدیه فی ال main واین قیمته مره خلالها

* ال local static variable یحافظ علی قیمته زی ماقولت بیتحفظ فی ال Rom وبناند نسخه ال RAM

وکن مره بستدی الفانکشن هو بیکمل بقیمته القدیمره لائن زی ما هو مجوز مکانه متمسکش وحتی امروه
مره تانی مثلاً