التاريخ: / /

الموضوع: _____

Lec 7

Casting :-

Converting one datatype into another is known as type casting or, type-conversion.

هى عملية تحويل من داتا تايب لـ داتا تايب تانيه من نوع لـ نوع آخر
يعنى مثلا لو عاوز اخزن قيمة من نوع int فى قيمة من long العكس دى
كدا اسمها casting

Casting

implict Type Casting
The compiler will automatically change
أى أنها تتم من خلال الـ compiler ←
من دون أن يحس بيها الـ user

EXPlist Type casting
You can convert The Values From one Type to another
تتم من خلال الـ user ←
(type_name) expression

*Rules :-

a) All integer types to be converted to Float.
b) All Float types to be converted to double.
c) All character types to be converted to integer.

التاريخ: / /   الموضوع:

Notes :-

```
unsigned int Numberone = 55;
unsigned long int numbertwo = 66;

float Var = 3.14;
```

long int ← int

```
Numbertwo = Numberone;
```

* هنا نحن قيمة من نوع int في نوع تاني long int الـ قام بالعملية دي الـ compiler بس المستخدم مش حسيس بيها عشان لو جيت طبعت قيمة Numbertwo هلاقيها 55

unsigned int ← float

```
Numberone = var;
```

* لو جيت اطبع Numberone هلاقيها بـ 3 بس

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

وهنا حصل casting

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

* الـ Var من نوع float واحنا بنحاول نطبعه كـ integer الـ output هيبقا بالشكل دا

Var = 1610612736

وكدا هو هنا مقدرش يعمل casting

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

* هنا عكس الفات قيمة integer نحاول اطبعها كـ float الـ output هيبقا بالشكل دا ← Numberone = 0.00000

هنا برضه مقدرش يعمل casting

الموضوع: _____ التاريخ: / /

* كدا اتضح إن الـ compiler ساعات بيقدر يعمل Casting وساعات لأ الـ compiler بيعرف يعمل فيها Casting دا كدا اسمها Implict casting وإنما الـ compiler مقدرش يعمل فيها أدى إسمها Explist casting آنا الـ user بحاول اعمل Casting كـ

* عشان كدا في شوية Rules لازم نمشي عليها في عملية Casting

* الشكل دا بيوضح عمليات التحويل من الداتا تايب بين بعضها

* هنا نحول من حت لفوق نحول لحجم أكبر.

* لو حولت فوق لـ حت بيحصل data loss عشان بحاول أخزن حاجة تجمها كبير في حاجة حجمها صغير.

Truncation of the Fractional Part

long double
double
float
unsigned long long
signed long long
unsigned long
signed long
unsiged int
signed int
unsigned short
signed short
unsigned char
signed char

Casting Rules :-

١- لو أنا عندى compiler ال short, char فيها expression بيحولهم تلقائى لـ int أو unsigned int.

٢- لو أنا عندى معادلة فيها operand من نوع long double ← باقى ال operand بتحول long double والناتج كمان بيتحول لـ long double

٣- نفس الكلام لو أحد ال operand من نوع double ← الباقى بيتحول لـ double والناتج أيضًا بيتحول double

٤- يطبق نفس الكلام مع ال float أيضًا

٥- لو كانه أحد ال operand ← unsigned long int/long ← باقى ال operand بتتحول unsigned long int/long والناتج أيضًا بيتحول.

٦- لو كانه أحد ال operand ← long int/long.
وال operand التانى من نوع ← unsigned int/long
يعنى لو واحد signed والتانى unsigned

← لو unsigned int اقدر احوله لـ signed long int فالحالة هيحصله عملية implict conversion long int وبقا long int والناتج هيبقا

← لو العملية السابقة دى فشلت فالـ compiler هيحولهم لـ unsigned long int والناتج أيضًا من نفس النوع

٧- لو أحد الـ operand ← long int ← الكل هيتحول لـ long int والناتج أيضًا

٨- لو أحد ال operand ← unsigned int هيطبق نفس الكلام.

```
unsigned int Numberone = 7;
unsigned int NumberTwo = 2;
unsigned int Result = 0;

Float Resultf = 0, Var1 = 24.5 , Var2 = 7.2

int main(){

Result=Numberone / NumberTwo;
Printf("Result = %i  - %f \n", Result, Result);
```

Result = 3 - 0.000000 ← شكل الخرج

* عشان قسم unsigned int على unsigned int فالناتج صحيح
فت نفس النوع أيضًا أما بالنسبة 0.000 هنا بحاول اطبع
قيمة int لـ Float

```
ResultF= (Float)Numberone / (Float)NumberTwo;
```

Explist casting

* كدا انا خليت قيمة Numberone من نوع Float ع العادلة
دي بس و أيضًا NumberTwo و بالتالي الـ ResultF من
نوع Float لو حبيت اطبع قصتها

```
Printf("ResultF = %i - %f \n", ResultF, ResultF);
```

ResultF = 0 - 3.500000 ← شكل الخرج

Float قيمة لـ
حاول اطبعها
int كـ

هنا ظهر تأثير
Explist casting الـ

Result= (unsigned int) Var1 / (unsigned int) Var2 ;

Printf("Result = %i \n", Result); }

Result = 3 ← تكملة الخرج

* هنا الـ Var2 < Var1 كانو float عشان عملنا explist casting
أصبحو unsigned int وبالتالي الناتج أصبح unsigned int

* يفضل إن انت تعمل عملية casting ومتسمحش الـ compiler أن يقوم بعملية الـ casting دي بدلها انت كـ user

---

integer Promotions in C :-

Some data types like char, short int takes less number of bytes than int, these data types are automatically promoted to int or unsigned int when an operation is Performed on them.

* بعض الداتا تايب زي char و short int بيحصلهم ترقية لـ int أو unsigned int تلقائياً من الـ compiler

EX:-

char Numberone =30 , Numbertwo =40, NumberThree=10;

char Result = (Numberone * Numbertwo)/NumberThree;
Printf("Result = %i \n", Result);

* من المعروف إن char ← 1 byte ولنفترض إن هنا signed يعني تمثل بشيل من -128 → 127 فعملية الضرب هنا كان الناتج 1200 والعدد دي أكبر من char فهنا حصل Promotion لـ int عشان يشيل قيمة 1200

```c
char Numberone = 0XFB;
unsigned char Numbertwo = 0XFB;

printf("Numberone = %c \n", Numberone);
printf("Numbertwo = %c \n", Numbertwo);
```

* هنا حصل promotion لإن أنا عايز عليهم if فحولهما int لـ Promotion

```c
if (Numberone == Numbertwo)
{
  printf("Yes \n");
}
else
{
printf("No \n");
```

output:-
Numberone = ✓
Numbertwo = ✓
No

← ليهم نفس binary في الميموري كـ value

```c
printf("Numberone = %i \n", Numberone);
printf("Numbertwo = %i \n", Numbertwo);
```

Hex	binary	decimal
0XFB =	1111 1011	= 251

output:-
Numberone = -5
Numbertwo = 251

* في عملية المقارنة فوق حصل promotion لـ int وبالتالي مبقوش نفس الـ value في الميموري عشان كدا طبع No

* يهمل البت الإشارة (الأخيرة) → unsigned → 251 ← 1111 1011

* يعمل حساب البت الأخيرة (الإشارة) → signed → -5 → 2's complement ← 0000 0101

التاريخ: / / الموضوع: _____

Memory Layout
Quick introduction

← أي بروجكت بعمله بيتكون من داتا وكود وبيكونو باخل الميموري

← المايكروكنترولر على الأقل بيتكون من نوعين من الـ Memory

RAM (SRAM)	Rom (Flash)
* بتحتوى على الداتا ال بنفذ عليها الـ operation	* بتحتوى على الكود الـ Processor هينفذه

- Basically, the memory layout of c program contains five main segments These are :-
  - Stack segment
  - Heap segment
  - Bss segment (Block started by symbol)
  - Data segment
  - Text or code segment

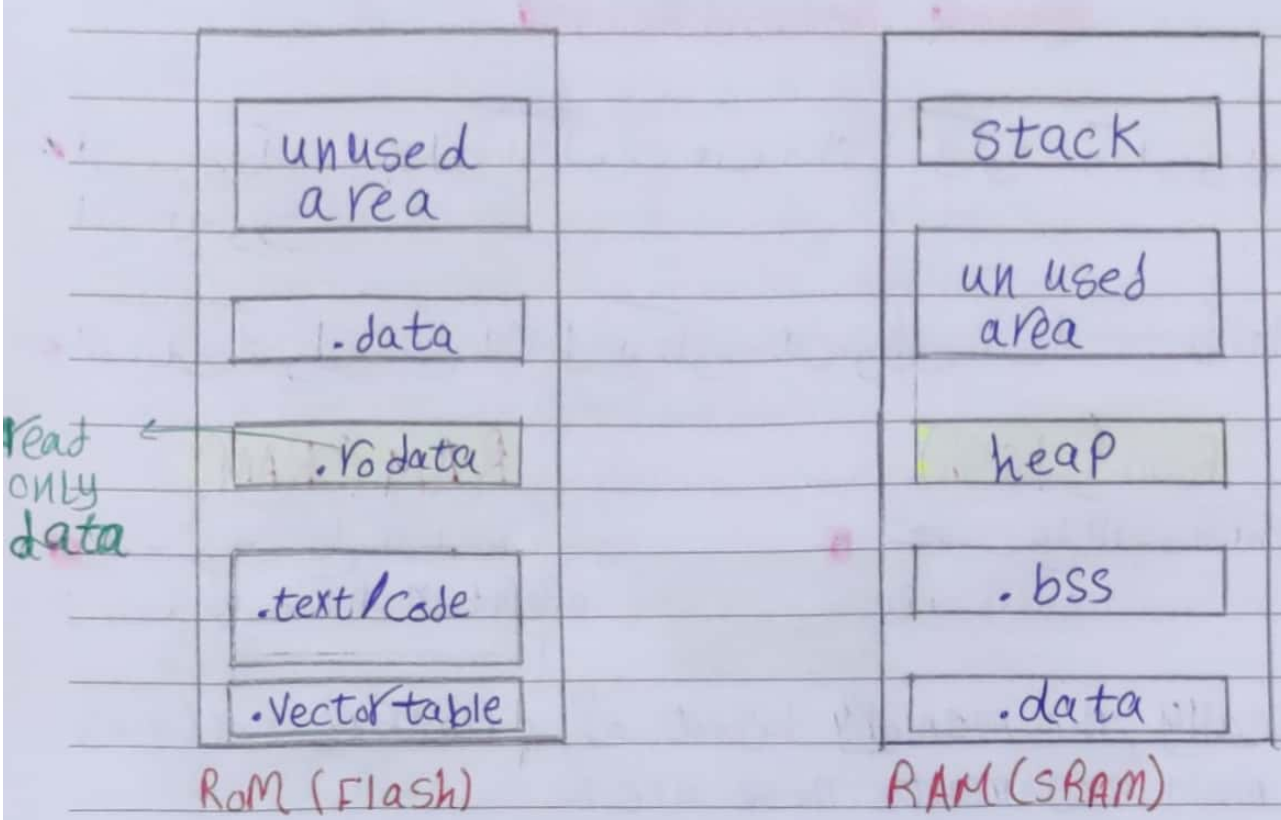
Each segment has own read, write and executable Permission.

* طبعًا أنا عندى أنواع كتيرة من الداتا، كل نوع ليه مكان مخصص لتخزينه فى الميموري.

التاريخ: / /

الموضوع: _____

• Structer of Memorey •



كل سكشن في الميموري بيتخزن في حاجه سواء كانت Ram , Rom

* unsigned int Numberone = 44;

* لو أنا عملت Global variable initialized زي دا ↑
و مستعملتوش مثلا أو عملت عليه أي عليه بعض الـ compilers
لتعتبره مش موجودة و ملوش مساحة في الميموري طب لو عملت عليه
أي أوبريشن مثل Numberone++ و
لو جيت أشوف مكانه في الميموري هلاقي منه نسختين
نسخه في Flash ← .data مساحته 4 byte
ونسخه في RAM ← .data مساحته 4byte
موجود منه نسختين عشان لو حصل قطع في الكهرباء الداتا متضيعش
وهيبقى موجودة في RoM وده ذاكرة بتحتفظ بالداتا حتى بعد انقطاع الكهرباء

التاريخ: / /　　　　　　　　　　　　　　　　الموضوع: ________

* الشغة الموجودة في ال RoM تظل ثابته وفي حالة حدوث أي عمليات على ال variable بتم على الشغة الموجودة في الRAM

ال startup code :-
هو عبارة عن كود بيـ run أول ميوصل كهربا لـ micro controller
وظيفته ← بيعمل كوبي لل data الموجودة في RoM وتخزنها في RAM ال data الموجودة في RAM وبيعمل Run قبل الmain

* unsigned int Numberone = 0 ;
* unsigned int Numbertwo;

ال Global variable initialized بـ صفر
و ال Not initialized
النوعين دول بيتخزنوا في ← RAM في bss.
أي حاجة جوا ال bss. قيمتها بصفر ال على كلها هو
ال startup code ← بيعمل قبل ال Run ال main

← يفضل يعني لو هتعمله بصفر شيل الصفر علشان بعض ال compilers بتعتبر ال 0 قيمة وبتضيع مساحة

* const unsigned int numberone = 99 ;

ال constant Global variable initialized بـ Value
بيتحجز له مساحه في RoM ← في rodata. عشان كدا مش بقدر اعدل فيه

.text/code → RoM
* السكشن دا بيتخزن فيه الكود زي الفانكشن ، فانكشن ال Main
و هنا مش قصدي على ال variables

أما بالنسبة لـ section → heap اللي في الـ RAM
بستعملها في حاجة اسمها dynamic memory allocation
يعني لما أجي أحجز مساحة أثناء الـ Run time

أما بالنسبة لـ section → stack
* كل الأنواع الجاية دي بيتحجزلها مكان في الـ stack

Local Variable Initialized
Local Variable Not Initialized
Local Variable Initialized to 0
Constant Local Variable Initialized with Value

* عشان لو أنا كامل Variable ← Not Initialized ← بيبقى
عشان متخزن في الـ stack وبالتالي garbage value
كان ممكن يكون في داتا سابقة ومتمسحتش

* الـ constant local variable اقدر اعمل فيها بطريقة
غير مباشرة عشان متخزن في الـ stack

* والـ local variable اقدر اغيرفيها بطريقة مباشرة

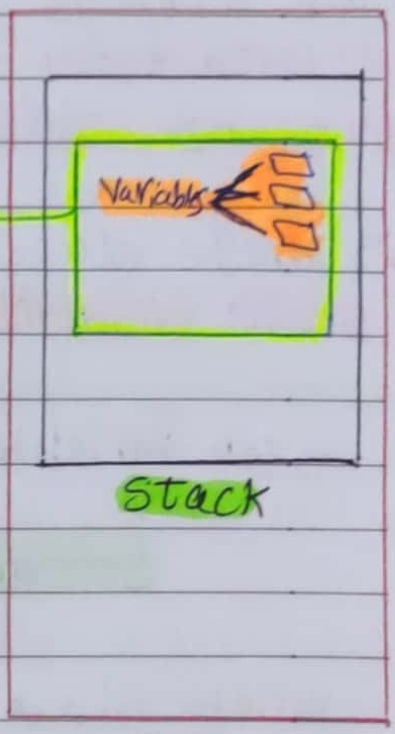
* لو أنا عندي فانكشن بالشكل دا

```
unsigned int Get_summing (int Num1, int Num2)
{
unsigned int Number1;
unsigned int Number2;

return (Num1 + Num2);
}
```

```
int Main(){
unsigned int Result = GetSumming(2,3);
Printf("Result = %i\n", Result);
```

* أول معمل Call لـ فانكشن في الـ Main بيروح على RAM في جزء الـ Stack ويحجز مساحة للفانكشن بالشكل دا ويخزن كل الداتا الموجودة بالفانكشن من داتا لـ Local Variables وأيضًا حتى لو كانوا const بيتحجزلهم مساحة Stack Frame ويبقى ليهم Mark كدا إن دول constant ومينفعش اغيرة قيمتهم بطريقة مباشرة



* قبل معمل Call للفانكشن كل داكمشن موجود في الـ Stack وقت مبعمل Call كل دا بيبدأ يتحجزله مساحه

* لما الفانكشن تخلص وظيفتها ونعمل Return كل دا بيتمسح تلقائيًا من الـ Stack والـ Stack Frame بيتمسح أيضًا

* لو عملت Call للفانكشن تاني هيبدأ يعمل Allocation لـ Parameter تاني وهيحصل لـ Stack Frame وكذلك الـ Variables اللي جوا الفانكشن

التاريخ: / / الموضوع: _____

Storage classes

- Now you will learn about the scope and lifetime of variables

- When you create a variable in C, then two things are always attached with variables:-
1- Data type
2- storage classes

- Storage class decides the extent (life time) and scope (visibility) of the variable in the program.

lifetime :- هو الفترة اللي بيفضل فيها الـ Variable موجود

Scope :- Global / Local

Global:
* اقدر اشوفه في أي مكان في البرنامج وأي حد يقدر يشوفه
* أي Variable مش موجود داخل {} في الحالة دي يعتبر Global Variable

Local:
* هو عبارة Variable داخل {} أي حد يقدر يشوفه جته داخل {}
* أي Variable متعرف داخل {} يعتبر Local Variable
* مش هقدر اعمل اكسيس ليه خارج {}

* يفضل الـ Local Variables تكون في بداية الـ function

التاريخ: / /

الموضوع: Local

* ينفع نبقا عندى اتين Variable بنفس الإسم فى اتين Function مختلفين وكل Variable بتتعمله اكسيس داخل {} فقط

* مينفعش نبقا عندى اتين Global Variable بنفس الإسم هيطلعلى ايرور Redefine عشان داخل Global والكل شايفه فلازم يكون واحد فقط

* مينفعش نبقا عندى اتين Local Variable بنفس الإسم داخل {} ورا بعض ذى كدا

```
int main(){
unsigned int Numberone=55;
unsigned int Numberone=66;
}
```

هيطلعلى ايرور Redefinition

* ينفع نبقا عندى Nested scope يعنى scope داخل الـ scope

```
{
  {
    لو انا هنا
    اقدر اشوف اى حاجه فوق لحد هنا
  }
}
```

* لو انا داخل الـ inner scope اقدر اشوف أى حاجه فوق داخل الـ outer scope

* يطبق نفس الكلام أيضا لو عندى scope داخلى كمان داخل الـ inner scope

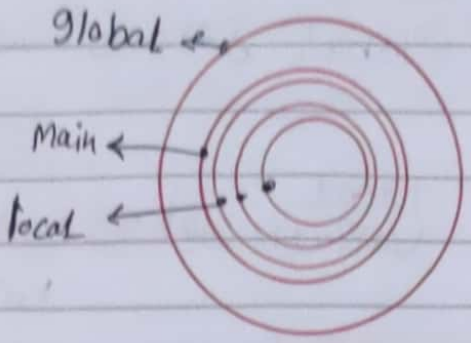
* ينفع نبقا عندى اتين Local Variable بنفس الإسم بس فى scope مختلف

* لما أجى استخدم Variable داخل Function دايما بيشوف الأقرب ليه

* أول مبخرج من {} أي Variable كان داخل {} هيتمسح من الميموري

* مجرد مبخرج من الـ main أي شيء باخلها هيتمسح

* الموضوع أشبه بالشكل دا



Life time

static

* يقصد بيه الـ global الـ variable هيفضل موجود طول عم البروجرام بيرن

automatic

* يقصد بيه الـ local في لحظة ما بيتمسح من الميموري على حسب الـ scope

auto (Automatic) :-

- The variables 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 particuler block only

الموضوع: التاريخ: / /

- 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.

```
unsigned int numberone =35;

auto unsigned int numbertwo=33;
```

- أي local variable ← by default هو ← automatic

* ممكن اكتب كلمة auto قبل ال variable وممكن اشيلها عادي مش هتأثر وعادةً مش بنكتبها

* مينفعش اعمل auto ع ال global هيطلع ايرور ال auto بتبقا local فقط

- register :-

```
register unsigned int numberone =55;
```

* كدا انا بقترح على ال compiler إنه يخزن ال variable دا في الميموري الموجودة داخل CPU وال compiler هو اللي بيحدد يتخزن ولا لا في اللحظة اللي يتنفذ عليه operation ما

* بعمل كدا عشان أوفر ال time بتاع عملية ال copy اللي بتاخدها من ال RAM لل CPU

* الـ register بستخدمها مع local variable فقط و مبتفعش اعلها مع global

* لو الـ compiler ملقاش مساحة في الـ CPU بيروح يخزنه في الـ stack

* بستخدمها مع الـ for عشان لو عندي variable بعمل عليه عمليات كتير فأنا بعمله register عشان أوفر في الوقت والتايم يبقى أسرع في عملية التنفيذ

* الـ register الموجودة داخل الـ CPU مقدرش أكسس العنوان بتاعها وبالتالي لو عندي variable ← register وحاولت أجيب عنوانه هيطلعلي error ← pointer مثلا

* مبتفعش اعمل local variable register واجيب عنوانه لو حاولت اجيب عنوانه هيجيب error عشان نحاول نجيب عنوان لحاجة ملهاش عنوان

extern :-

* بستعملها لو عايز اعمل declare function أو declare variable

```
extern unsigned int Numberone;
```

* كدا انا عملت declare لـ variable اسمه Numberone من نوع unsigned int بس متحجزلوش مساحة في الميموري لو جيت اعمل build و Run مش هيرن ← linker error

```
extern unsigned int Numberone = 5;
```

* كدا أصبح definition والحجز له مساحة وكلمة extern ملهاش لزمة

الموضوع: _____ التاريخ: / /

* لستخدم ال extern لو أنا عندي Multi Files

main.c

```
extern unsigned int var;
#include "motor.h"
```

* هنا كان عندي variable في file ← motor.c وعاوز استخدمه في ال main.c ممكن اعمله بطريقتين :-

① معمله declare عن طريق إن استخدم extern قبل ال variable وبدأ اقدر استعمله في ال main.c

② ممكن اعمل ال declekation في قايل motor.h واعمل include لل .h في ال main.c وبفضل إن استعمل الطريقة دي

Motor.c

```
#include "motor.h"

unsigned int var
```

* ممكن اطبق نفس الكلام مع الـ function ممكن استخدم extern وممكن ← مش متفرق

Motor.h

```
extern unsigned int var;
```

التاريخ: / / الموضوع: _____

Static

* بستخدمها مع الـ Function أو الـ Variables ← Local / Global

* With Global :-

```
Static unsigned int Var = 55;
```

* كدا انا بقول ان داGlobal Variable static يعني مش هقدر اشوفه خارج الـ File الهوفيه يعني أصبح File scope على عكس لو أنا شلت الـ static كله شايفه لو عايز استعمله في فايل غير الهو فيه بعمل declaration ← extern ← واستعمله عادي

Project scope → File scope
↳ يعني الفايل اللي فيه هو بس اللي يقدر يـ access عليه

* With Function :-

* نفس الكلام أيضًا اللي يطبق على Global Variable يطبق على Function لو أنا كتبت static قبل الفانكشن كدا محدش يقدر يعملها call غير في الفايل اللي هي فيه يعني أصبحت File scope

* With local :-

* لو أنا مثلا عندي فانكشن وجواها static Variable بالشكل دا

```
Void PrintMotorVar(Void)
{
static unsigned int Var1 = 55;
// codes
}
```

الموضوع: ________ التاريخ: / /

* لما اجي اعمل call للفانكشن ← هيعملها stack frame في الـ stack في الرام ويتحجزلها مساحة للكود الجواها بس static variable هيتحجزله مساحة في الـ RAM ← data section. وهيفضل معايا طول فترة الـ project.
والـ Flash ← data section.
يعني لما الفانكشن تخلص وتتمسح من الـ stack الـ static variable هيفضل في الـ RAM ← data (ROM → data) .data. وبكدا أصبح الـ lifetime الخاص بيه هو الـ lifetime بتاع الـ project كامل

IMPortant Notes

* لما اجي اعمل file.h مينفعش يكون فيه أي definition سواء كان variable defenition أو function defenition عشان لو عملت include ليه أكتر من مرة هيعملي error

* ينفع اعمل include لفايل .c مرة واحدة بس علشان لو عملت أكتر من مرة هيطلعلي error ← re defention ويفضل أساسا إن أنا معملش include لفايل .c بعمل include ← لـ .h فقط

* الفايل .h الخاص بالموديول اسمه ← Master interface header file

* لو أنا عندي variable في فايل .c مثلا وعايز استخدمه في الـ main بروح اعمل decleration للـ variable دا في الـ .h الخاص بالموديول واعمل للـ .h ← include في الـ main.c

التاريخ: / /

الموضوع: _____

* ال static بتلغي تأثير ال extern يعني لو أنا عندي Variable من نوع static في فايل c. مثلا وعايز استوفيه في ال main.c وجيت عملت declaration له في فايل h. وجيت استخدمه في ال main.c عشان هيرضى هيعمل error لأن ال static ← file scope وبالتالي مفيش غير طريقه واحده ← إن اعمل فانكشن getter في فايل c. بترجعلي قيمة ال Variable دا واعمل declaration لها في h. واقدر اعملها call في ال main.c بعد معملت include لفايل h. في main.c

* ال static function يفضل يكتب إن ال declaration الخاص بيها دايما في بداية الفايل c. وليس في فايل h.

* لو عندي Global static variable في فايل c. وعايز اعدل في قيمته واعمل فايل تاني مثلا ولكن main.c ← يعمل فانكشن اسمها setter بياخد ليها ال Value في Variable و ال Variable دا اكتبه في ال static variable

* ال Local static variable بيحافظ على قيمة ال Variable يعني مثلا لو عندي Local static var داخل فانكشن وأنا بعمل اوبريشن على ال Variable دا مثلا ++ وجيت عملت call للفانكشن أكتر من مرة مثلا أول ما هيتم حجز مكان في ال stack هو stack frame وفيه كل محتويات الفانكشن معادا ال static variable هيتم حجز ليه مكانين ← RAM ← data section. وبمجرد الإنتهاء من الفانكشن هيحصل destroyed stack frame بس ال static variable هيفضل زي ما هو موجود في ال RAM ← data section وهيحتفظ بآخر قيمة اتخزنت فيه