Eng. Ahmed Assaf Embedded Systems 12.12.2018

#define Func(x) do{

#if (x)==

#else

}while(0)

error

لازم الـ x = 0

macro name متغير مش

Data types

Primitive

char
int
float
double
void

pointer to void

Func Return

Func Argu.

* Sign Modifier ANSI "C" → default sign modifier is a compiler dependent.

Signed → char
unsigned → int

Signed char → Pos & Neg. numbers
unsigned char → Pos.

؟ unsigned ولا signed Char سؤال: هوا

Ans. it's compiler dependent.

Q if I write:
Signed char = 200; → هيخزنها بالرقم المقابل ليها بالـ range بالـ negative
unsigned char = -1; → هيخزنها 255

if I write:
unsigned char x = -1;

if (x == -1) → "Temporary variable"
{ (معظم الـ compiler بتعتبره لكن في النهاية signed)
(هي compiler dependent)
}

Implicit casting: يعني الـ compiler هو اللي قرر هو هيعمل الـ casting إزاي.

Rule ← متستخدمش variable من غير sign modifier.
Rule ← unsigned modifier بيزود الـ range علشان كده لو أقدر أستعمل unsigned

(*) Size Modifier
Short → int
long → int, double
long long → C99

C89 → الكود المكتوب عليه ممكن يعمله compile على C99 لكن العكس لأ، علشان كده أفضل استعمال C89

char	is not ~~more~~ less than 1
short int	" " " " 2
int	" 2 (or) 4
long int	" 4
float	" 4
double	" 8
long double	" 10

Sizeof( ) → operator not function
↓ Data Type ↘ variable

Size of our Data Types → Environment dependant
- Processor
- tool chain
- OS

Environment dependant كل حاجة قولنا عليها هي في الحقيقة Compiler dependant

Typedef: Type definition

- جملة typedef مش بتاخد مساحة في ال memory هي بتعلم ال compiler datatype جديد

Syntax

typedef [old type] [New type]

typedef (unsigned int) (uint;

(old → unsigned int, new → uint)

- مفيش أي فرق إنك تعمل New datatype using typedef or #define

↓

لكن في حالة مش هتنفع فيها

↓

```
typedef int* Pint;
#define Pint int*;
```

Pint x,y; ← في حالة ال define ال x هتكون Pointer وال y هتكون int

أما ال typedf هتكون ال x و y هيكونوا pointers عادي

#define بتتعمل بال preprocessor
#Typedef " " Compiler

STD_Types.h

typedef unsigned char — u8
typedef unsigned short int — u16
u32
s8
s16
s32
f32
f64

#ifndef ... #endif — guarded header file

Storage modifiers → location, life time, Scope

auto, register, *static, *extern

auto, register → 10 → Datatypes only

static, extern → comes with → Datatypes + Function

auto		register	
Location	RAM	GPR or RAM	
Lifetime	Local { } / Global All Time	{ }	All time
Scope	{ } / All Project	{ }	All project

Static		extern	
Location	RAM	No memory	
Lifetime	All time	{ }	All time
Scope	{ } / All file	{ }	All Project

① auto

```
auto u32 x=10;
Void Func void
{
  auto u32 x=20;
  printf("%ad", x);
}

Void main(void)
{
  auto u32 x=0;
  printf("%d", x);
  Func();
}
```

* لو في 2 variables فيأخذ الأقرب
auto with the same name But with different scope

* C99 معفاش auto وهي الطريقة الوحيدة اللي موجودة في C89 وميش موجودة في C99 فهتعمل compilation error وبالتالي.

Rule → It is forbidden to use auto with variables

Register

كأني بقول للـ Compiler أنا بقترح عليك إني أحفظ الـ var ده عندك في الـ GPR ومن حقك الـ Compiler إنه يوافق أو يرفض.

Rule ← كلما كان حجم الـ variable أصغر كلما كانت احتمال إنه يتحط في GPR أكبر

Rule ← كلما كان تكرار الـ access على الـ var أكتر كلما كانت احتمال إنه يتحط في GPR أكبر

* أوقات بيحصل Conflict في بعض الـ Compiler لو اتحطت Global variable مع register

فعلشان كده الأولوية والأفضلية إنه register يكون مع Local variables لأنه الـ local هيستعمله مرة واحدة ويسيبه وكده ممكن 5 local يستعملوا register واحد لكن الـ Global طول ما هيسيب

اسم الـ GPR بستعمله علشان أعمل access عليه علشان كده السطر ده غلط

```c
register u32 x=0;
u32* ptr=&x;
```

غالباً هيحفظه في الـ RAM عادي وبكده هيعتبر كأني متكتبتش كلمة register أصلاً.

File1.c

```c
u32   x=100;

void  Func(void);
void  Func3(void);
void main(void)
{
    Func();
    Func3();
}
```

File2.c

```c
extern  u32 x;

void  main(void)
{
    Printf("%d", x);
}
```

مجرد صورة علشان السطر ده يعدي الـ Compiler ويوصل لمرحلة الـ Linker

compiler will produce "Symbol table"

x	u32	provided
main	function	provided
Func	function	needed

"Symbol table"

Func	function	provided
x	u32	needed

*Note * لو عملت Compile & run لـ File1.c (الموجود لوحده من غير file2.c) هيطلع .obj file لكن مش هيطلع .exe File علشان كده مش هيطلع Linker error

* لو مسحت سطر الـ extern u32 x // هيطلع Compilation error اسمه 'x' undeclared

* لو مسحت سطر الـ u32 x=100; // فده هيطلع Linker error اسمه undefined reference to 'x'

File3.c

```c
u32 x=50;

void func3(void)
{
    printf("%d",x);
}
```

Symbol table

x	u32	Provided
func	function	Provided

* الـ error إن الـ Linker هو Multiple definition of 'x'

* لو كتبت قبلها السطر static u32 x=50; جوه الـ File3.c

فكده الـ x مستخبية جوه File3.c ومش مكتوبة جوه الـ Symbol table

Note: لو كتبت كلمة extern لـ var. وعملت init. في نفس السطر فيحصل compilation error أو هيعمل neglect لكلمة extern وبالتالي هيكون var. عادي