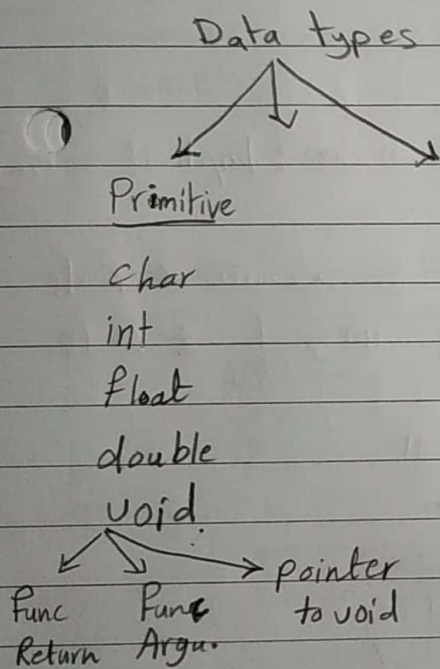


`#define Func(x) do {`
`# if x == 1`
`//`
`# else`
`//`
`} while(0)`

error → `do { x = 1; }`
 macro name `do { x = 1; }`



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

Signed → char
 unsigned → int

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

? unsigned & signed char → `signed char` & `unsigned char`

Ans. it's compiler dependent.

Q if I write:

Signed char = 200; \rightarrow ^{بالقيمة} الرقم المقابل لها
 unsigned char = -1; \rightarrow ²⁵⁵ \rightarrow ^{مخزن} 255

if I write:

unsigned char x = -1;

if (x == -1) \rightarrow "Temporary variable"
 { \rightarrow معظم ال Compiler يتوقع
 } signed لكن في النهاية
 Compiler \rightarrow dependent

Implicit casting: يعني ال Compiler هو اللي قرر هو هيجعل ال Casting ازاوي.

Rule \rightarrow متضمنة Variable من غير sign modifier.

unsigned \rightarrow Rule \rightarrow بينود ال range علشان كده لو اقدر \rightarrow ^{معدل} unsigned modifier

(*) Size Modifier C89 \rightarrow الكود المكتوب عليه
 Short \rightarrow int ممكن يوصله compile على C99
 long \rightarrow int لكن العكس لا علشان كده
 long long \rightarrow C99 افضل با استعمال C89

Char	is not 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 Type → Environment dependant
 { Processor
 tool chain
 OS

Environment dependant كل حاجة قوليها Compiler dependant

typedef: Type definition

typedef عمل بي بي ip له في ال memory
 compiler ال datatype

Syntax

typedef old type New type
 ↓ ↓
 typedef unsigned int (uint);

New datatype - عمل اى فرق في ال
 using typedef
 or #define

اكن في حالة من منع فيها
 ↓

typedef int* Pint;
 #define Pint int*;

int x, y; x و y هتكون Pointer وال

اما ال typedef هتكون ال x و y هتكون pointers فادي

#define بال preprocessor
 Compiler " " #typedef

STD_Types.h

```
typedef unsigned char    48
typedef unsigned short int 416
                           432
                           58
                           516
                           532
                           632
                           664
```

#ifndef آکٲٲ آکٲٲ آکٲٲ guarded آکٲٲ header file
#endif

Storage modifiers → location
→ life time
→ Scope

auto register *Static *extern
↓ ↓
↓ → Datatypes only comes with ↓ → Datatypes + Function

auto		register	
Location	RAM	GPR or RAM	
Lifetime	{ Local } Global All Time	{ }	All time
Scope	{ }	{ }	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("%d", x);  
}
```

```
void main(void)
```

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

* لو في 2 Variables فيها نفس الاسم
auto with the same name
But with different scope

* C99 معناه auto ودي الطابة الوليدة الى موجودة في C89
ومش موجودة في C99 فموجود error وبالتالي .

Rule → It is forbidden to use auto with variables

Register

كأني بقول لل Compiler أنا بقص عليك راني أحفظ ال var ده عندك
في ال GPR او هن صفا ال Compiler انه يوافق أو يرفض .

Rule ← كلما كان مع ال variable أصغر كلما كانت احتماله انه يبقى في GPR أكبر

Rule ← كلما كان تكرار ال access على ال var أكثر كلما كان احتماله انه يبقى في GPR أكبر

* أوقات بيحصل Conflict في بعض ال Compiler لو استخدمت Register مع Global variable

فعلشان كده الأولوية والأفضلية لانه register يكون مع Local variable لأنه ال Local variable مش ممكن ال register واحد لكن ال Global مش ممكن

File3.c

u32 x=50;

void func3(void)

{

printf("%d", x);

}

Symbol table

x	u32	Provided
func	function	Provided

Multiple definition of 'x' * error في الـ linker

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

فكرة الـ x مستخدمة في File3.c وحس مكتوبة في الـ Symbol table

Note: لو كتبت كلمة extern لـ var. و init. في نفس الـ compilation ~~error~~ أو neglected ~~error~~ و بالتالي هيكون الـ var. عادي