①

C Basics

* Compilation process :-

⓪ Main.c — source code (.c) (.cpp)

① #, .h — PreProcessing

↓

② .i — Macro (.i) Compilation (gcc) (g++)

↓

③ .s — Assemble (Assembly code)

↓

④ obj , .o — static Library ← Linking — Linker (ld) (.o)

\+

↓

⑤ .lib — Execution (.exe)

⑥ .exe

Notes :-

(.i) ← ده بعد مرحلة ال preProcess بيتحط مكان اى فايل (.h) زى
مثلا stdio.h وبعدين بياخده gcc ويعمل compilation للكود
او ال instructions بتاعتك وبعدين بياخده (.s) لل Assembler
يطلع ال binary الخاص بكل instruction وبعد كده ياخد
يشوف (.lib) عشان يعمل executable file
يعني ال Linker بياخد (.obj) ويضيف الحاجات اللي محتاجها ال (.lib)
ويجمعهم مع بعض وكده يبقى ده ال executable file

compiler + Linker + C. Libs + Binary utilities ← Toolchain

native

* install -> intel

I.C -> I.exe => run on intel

MinGW -> Native tool chain

cross

Atmel studio -> intel

(.c) -> .bin -> Burn on Flashmemory

Atmega

* Variable Name :-

← لازم ميكونش reserved key زي void, int

← لازم ميكونش فيه spaces

← تعرف تحط رقم بعد ال variable ومينفعش قبله

* Comments :-

← فيه طريقتين :-

(1) // ....

(2) /*

... */

* Data Types :-

→ Basic Types or Primitive :-

- Integer values
  - signed ±
  - un signed +
- Real values :- Fractions, ex, int فيها جزء

→ Derived :-

- Arrays
- structure
- Union
- Pointer

→ user defined :-

← انت اللي بتعرفه بنفسك :-

(1) enum

(2) Typedef

(3)

* Integer values :-

unsigned Integer :-

Data type	Major type	size (Bytes)	Precision	Range
char	Integer	1	1	-128 to 127
unsigned char	Integer	1	1	0 to 255
short	Integer	2	1	-32,768 to 32,767
unsigned short	Integer	2	1	0 to 65,535
* int	Integer	4	1	-2,147,483,648 ->
* unsigned int	Integer	4	1	0 -> 4,294,967,295
Long	Integer	4	1	-2,147,483,648 ->
unsigned long	Integer	4	1	0 -> 4,294,967,295
long long		8		-9,223,372,036...
unsigned long		8		0 ----> 18,446...

Unsigned Integer

(Size in bytes) -> no's

$0 >>> (2^{} - 1)$

signed Integer

(Size in bytes - 1) (Size in bytes - 1) -> no's

$-(2^{}) >>> +(2^{} - 1)$

For Example :-

Unsigned short -> 16 bits

- The minimum value is $(0000\ 0000\ 0000\ 0000)_2\ (0)_{10}$
- The maximum value is $(1111\ 1111\ 1111\ 1111)_2 = (65535)_{10}$

Date: \ \

* floating point types :-

Type	storage size	Value Range	Precision
float	4 byte	1.2 E-38 to 3.4 E+38	6 decimal places
double	8 byte	2.3 E-308 to 1.7 E+308	15 decimal places
long double	10 byte	3.4 E-4932 to 1.1 E+4932	19 decimal places

* complement

5 = 0 0 0 0 0 1 0 1
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ } complement digits
1 1 1 1 1 0 1 0
+ 1 } Add 1

-5 = 1 1 1 1 1 0 1 1

* bool :-

→ there was no boolean in Ansi C
u can include this library to use it <stdbool.h>

int / float → float يكسب

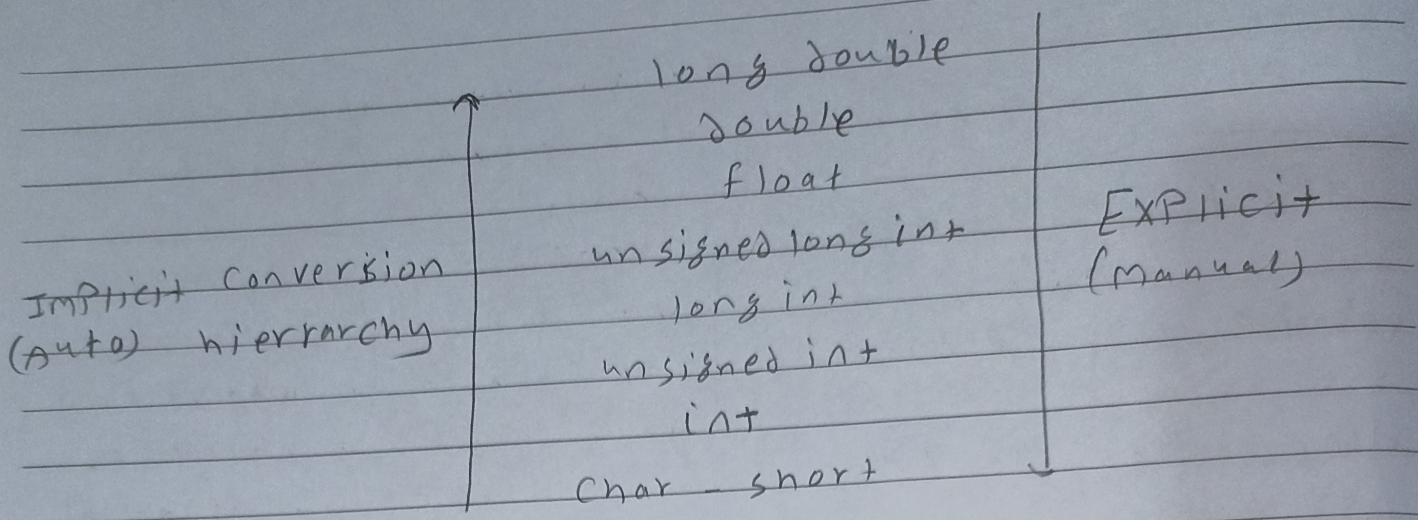
* Type conversion in C :-

→ Implicit Type conversion :- Automatic

→ Explicit Type conversion : Manual

(type) variable exi- (int) X

Date: \ \



* Hirachry of operations:-

Priority	operators
1st	*, /, %
2nd	+ -
3rd	=

* Printf() and scanf() (I/O):-

-> Printf and scanf used only in c Programming and you can not use These functions in Embedded c because you don't have a screen to Print the text or out Put on it.

Ex:-
```
#include <stdio.h>
int main(){
int testInteger;
Printf("Enter an integer:");
fflush(stdout);
scanf("%d", &testInteger);
printf("Number = %d", testInteger);
return 0; }
```

Date: \ \

Note:-

'\r', '\n'	makes new line
'\t'	inserts a tab
'\\'	prints '\'
'\''	Prints ''
'%d'	Prints or scans an (int) value
'%x' or '%X'	Prints or scans an integer value in small or capital Hex format
'%f'	Prints or scans a real (float) value
'%lf'	Prints or scans a real (double) value
'%u'	Prints or scans an (unsigned int) value
'%c'	Prints or scans a single (char)

* Math and logical expressions:-

=	Equal operator
+	Addition
-	subtraction
*	Multiplication
/	Division
%	(Mod) Ex 15%4 = 3
( )	Parenthesis
++	Increment
--	Decrement
|	OR
&	AND
^	XOR
~	NOT
>>	shift Right
<<	shift Left
+=	self Addition Ex: X+=2

(7)

Date: \ \

* Prefix, Postfix:-

++x

go to memory
&x
→ x = x+1
→ load the final value

x++

Print x first
Then go to memory
and Increment x by one

→ To set specific bit:-

R1 = 1 << n

→ To clean specific bit:-

RR = ~(1 << n)

→ To toggle specific bit:- if 0 -> 1 if 1 -> 0

R ^ = 1 << n

* Conditions:-

True 1 → any number except 0
false 0

* if Statement:-

```
if ( /* if condition */)
{
  // if body
}
else if ( /* else if condition */)
{
  // else if body
}
else{
 //else body
}
```

Date:

* **Inline condition:-**

(condition) ? true : false;

* **switch statement:-**

```
switch (/* switch expression */)
{
    case /* case value */ :
    {
        //case body
    }
        break;
    ...
    ...
    case /* case value */:
    {
        // case body
    }
        break;
    ...
    ...
    default:
    {

    }
```

→ case must be int const

-> switch is more faster than if.

→ if condition has higer condition capapility.

* مينفعش تعمل case لأي حاجة فيها fraction زي مثلاً float

أو تكرار الـ case مرتين كدا

(9)

Date:

* for statement :-

```
for (/* intiation*/; /* condition*/; /* increament*/)
  {
    // for body
  }
```

carful :

```
for( ; ; );
 {
 }
```

this for do Null (nothing)

* while statement :-

```
while (/* condition */)
  {
    // while body
  }
```

Important :

break statement is used to exit from any loop type.

* do...while statement :-

```
do
{
  // do...while body
}
while (/* condition*/)
```

Date: \ \

* goto statement:- tells the Program where to jump

```
// C statement
labelname:
// C statement
// C statement
goto labelname;
// C statement
```

OR

```
// C statement
goto labelname;
// C statement
// C statement
labelname:
// C statement
```

* break statement:-

- بتقطع بيها أي loop

* continue statement:-

- بترجعك لأول (أول) الـ loop

* Nested loop:-

- loop جوا loop لحد ما تخلص

Date:

(11)

Quiz 1

CBasics tricks

① unders_ في تسمية ال variable يمكن استخدامها في أي مكان في الاسم ولكن لا يمكن أن تبدأ برقم ولا تكون keyword

② Evaluate the following expressions

ⓐ → g=big/2+big*4/big-big+abc/3; (abc=1.5, big=3, assume g to be float)

ⓑ → on = ink*act/2+3/2*act+2+tig; (ink=3, act=2, tig=3.2, assume on to be an int)

ⓒ → s=qui*add/4-6/2+2/3*b/god; (qui=2, add=4, god=3, assume s to be an int)

ⓐ big/2 → 3/2 → int/int → implicit → 1

big*4 → 3*4 → int*int → 12

12/big → 12/3 → int/int → 4

4-big → 4-3 → int-int → 1

abc/3 → 1.5/3 → float/int → float → 0.5

1+1+0.5 → int+int+float → float → 2.5

∴ g = 2.5

1+12/big-big+abc/3

1+4-big+abc/3

1+1+abc/3

piriority for float

~~~~ for float

ⓑ ink*act → 3*2 → int*int → int → 6

6/2 → int/int → int → 3

3/2 → int/int → int → 1

1*act → 1*2 → int*int → 2

3+2+2+tig → int+int+int+float → 3+2+2+3.2 → float → 10.2

on is int ↵

so, on = 10

6/2+3/2*act+2+tig

3+3/2*act+2+tig

3+1*act+2+tig

3+2+2+tig

ⓒ qui*add → 2*4 → int*int → int → 8

2/3*b → int/int*int → int → 0

8/4-6/2+2/3+b/god

2-3+0/0 = -1

∴ s = -1

Two tricks → int/int = int, float/int = float

→ operation priorities
~~~~

Date:

③
```
int i=2, j=3, k, l;
float a, b;
k = i/j*j;
l = j/i*i;
a = i/j * j
b = j/i * i
printf("%d %d %f %f", k, l, a, b);
```

Two tricks → int / int = int float / int = float
→ operation priorities

i/j*j → i/j → int/int/int → 2/3 → 0 → 0*j → 0*int → int → 0
∴ k = 0

j/i*i → j/i → int/int → int → 3/2 → 1 → 1*i → 1*int → 1*2 → 2
∴ l = 2

i/j*j → i/j → int/int → int → 2/3 → 0 → 0*j → 0*int → 0*3 → 0
but a is float ∴ a = 0.0

j/i*i → j/i → int/int → int → 3/2 → 1 → 1*i → 1*int → 1*2 → 2
but b is float ∴ b = 2.0

④ Programs are converted into machine language with the help of
(1) An interpreter → يطبق execute يبدأ ويشتغل run
(2) A compiler → Binary يطلع
(3) an operating system → ؟ toolchain اللي installed
(4) None of the above

⑤ The real constant in C can be expressed in which of the following forms
(1) fractional form only
(2) Exponential form only
(3) ASCII form only
(4) Both fractional and exponential forms
real number
long double, double, float يعني
Ex 1.5 or e^{-1}

⑥ which of the following statement is wrong:
(1) mes = 123.56;
(2) con = 'T' * 'A';
(3) this = 'T' * 20;
(4) 3+a = b → data اللي جوه a الى var على الـ

(13) Date: \ \

(8) Which of the following shows the correct hierarchy of arithmetic operations in C?

(1) (), **, * or /, + or -
(2) (), **, *, /, +, -
(3) (), **, /, *, +, -
(4) [(), / or *, + or -]

** -> means Pointer to Pointer

(10) The expression a = 7/22*(3.14+2)*3/5; evaluates to

(1) 8.28
(2) 6.28
(3) 3.14
[(4) 0]

Two tricks -> int/int=int Float/int=float
-> operation priorities

7/22 -> int/int -> int -> 0

0*0/0 -> 0

(11) assume that a is int with 2 bytes

The expression, a=30*1000+2768; evaluates to

(1) 32768
[2) -32768]
(3) 11340
(4) 0

Tricks -> signed int range for 2 bytes

-2^{8n-1} to $2^{8n-1}-1$ ... range ... signed ...

n -> number of bytes

So the range between -32768 to 32767

... 2bytes ... Max ... expression ...

1000 0000 0000 0000 ... Binary ...

... (-ve) ... (+ve) ...

... 0111 1111 1111 1111 + 1

... (-ve) ... (+ve) ... 1000 0000 0000 0000

(12) The expression $x = 4 + 2 \% (-8)$ evaluates to

(1) -6

[(2) 6]

(3) 4

(4) None of the above

ماهو % دي عامله (بيعملها الـ mod اللي في الرياضيات) ← trick

في الـ %

+ve % -ve = +ve

-ve % +ve = -ve

-ve % -ve = -ve

يعني الـ % هيستمر بـ الـ (-ve) إلا لو كانت في البسط

(13) Assume that the size of the char is 1 byte and negatives are stored in 2's complement form.

```
#include <stdio.h>
int main()
{
    char c = 125;
    c = c+10;
    printf("%d",c);
    return 0;
}
```

* يعني الـ range بتاع الـ 1 byte من -128 الى 127

يعني ده معناه إن الـ c عدى الـ range

c = 125 + 10 = 135

c ← 1000 0111 (يتعبر عن -ve)

أي واحد من الشمال فاهو يبقى -ve

1000 0111

0111 1001

121 ← c

ده معناه إنّ c = -121

* طريقة تانية ممكن نجيب الـ range من -128 → 127 ونبعجر

مع غير الـ -ve هيبقى 256 ونطرح الـ 256 من الرقم ده الـ expression

هيبقى 135-256 اللي هو هيساوي برضه -121

(14) Predict the output, Assume that the size is 2 bytes.

```
unsigned x = -1;
int y = ~0;
if (x == y)
    printf("same");
else
    printf("not same");
```

الـ -1 ده معناه 1111 1111 1111 1111

يعني بـ الـ Hex ffff

الـ y = ~0 معناها إعدال y ومش 0 يعني

برضه هتبقى ffff

∴ 0xffff = 0xffff   x = y

فـ هيطبع same

Date: / /

* trick -> ASCII with octa number

(16)
```
char a = '\012';
printf("%d", a);
```

* الـ (\0) العلامة دي معناها octa
يعني هنا هو عايز يجيب الـ Dec للرقم 012 اللي هو octa وبعدين يجيب
الـ ASCII في من الآخر هيطبع الـ ASCII اللي هو 10

(17) Predict the output of the below Program.

```
printf("%d", 1 << 2 + 3 << 4);
```

* الـ add قبل الـ shift

عندنا قاعدة علشان تعمل left shift لأي رقم
هي الرقم المضروب ← 2^N * الرقم   ولو يمين shift هتبقى N >>> الرقم / 2^N

∴ 1 << 5 << 4

1 << 5 = 1×2^5 = 32

32 << 4 = 32×2^4 = 512

طيب دي الطريقة السهلة ← الطريقة الصعبة بقى
بتجيب الـ Binary للرقم اللي انت عاوز تعمله وتعمل الشيفت
وبعدين تحوله مرة تانية Dec

1 << 5 << 4

1 << 5

0000 0001

وعاوزين نعمل شيفت << 5 أرقام
يعني هتبقى 0010 0000 اللي هو برضه
32

32 << 4

هنا عندنا الـ 32 تحولها للـ Binary
0010 0000 وبعدين تعمل شيفت
يعني هتبقى 0010 0000 0000
الناتج 512 = 2^9

trick -> addition is highest Priority over shifting operators

Date: \ \

(18)
```c
int i = (1, 2, 3);
printf ("%d", i);
```

trick

* علامة الـ , هنا معناها إن أنا في الأول كانت بتساوي (1) , وبعدين (2) وبعدين (3) فهو هيطبع (3)

(19)
```c
int i = 5, j = 10, k = 15;
printf ("%d", sizeof (k /= i + j));
printf ("%d", k);
```

trick -> optimization tricks

* الـ size of بيجيب الـ size بتاع الـ Type يعني لو طلع (int) فهيطبع الـ size بتاعه اللي هو 4 ف هنا هيطبع 4 وبعدين 15

(20)
```c
int a = 10, b = 20, c = 30;
if (c > b > a)
    printf ("True");
else
    printf ("False");
```

-> Multiple comparison

Trick -> (a == b == c) evaluates in C programming (==) operates from left to right

Expression a == b == c is actually (a == b) == c

* الـ compiler هيروح يشوف c > b وده صح لو آه هيطلع (1) لو لأ هيطلع (0)
وبعدها هيروح للي بعده اللي هو هل 1 > a ولما تقارنها هيبقى لأ فهيطلع 0
يعني مرة الأولى فـ الـ compiler بياخد شرط واحد ، ناتجه ، بعدها يخش بيه الشرط اللي بعده فهو هنا في حالتنا هيطبع False
(c > b) > a

(21)
```c
int a = 0;
int b;
a = (a == (a == 1));
printf ("%d", a);
```

* هنا هيطبع (1) ليه؟ لأن أصلاً بيقارن
بيقول هل (a == 1) فهو بيقول لأ يعني (0)
وبعدين هيقول هل 0 == a فهو بيقول آه يعني (1) فهيروح يطبع (a) اللي هي بـ (1)

* Implicit & Explicit tricks

------ conversion is done programaticully.

(1) implicit
(2) Explicit
(3) other

* لازم انت هنا بتدخل ومش بيحولها Auto

* لو بتحول من Larger -> smaller
هو الـ Explicit

* الـ conversion اللي بيتعمل Auto هو الـ implicit

* الـ conversion اللي بيحول من Larger -> smaller هو الـ implicit

(30)

```
int a = 1;
int b = 0;
b = a++ + a++;
printf("%d %d", a, b);
```

- في الأول الـ a في الـ memory بقيمة صفر
يقول a++ دي معناه هياخد الـ a الموجوده
في الـ memory وبعدين هيعمل increment في الـ memory
يعني في الأول a++ 1 في الـ a
التانية الـ a في الـ memory هتبقى 2 وبعد كده هتـ 2+1 يعني الـ b=3
وبعدين هيخزن الـ a في الـ memory بعد الـ increment بـ 3 فهيروح يطبع
الـ b = 3 والـ a = 3

بس ده في compilers معينه و compilers تانية لأ في ده هيبقى
compiler dependent

يعني يعتمد على نوع الـ compiler فلو متقيدش في compiler dependent في
الاختيارات هتشتغل من الشمال لليمين ولو كانت بكتوبة جوه
printf هتشتغل من اليمين للشمال

(31)

```
int x = 10;
int y = (x++, x++, x++);
printf("%d %d \n", x, y);
```

* هنا لازم هيزودوا لما بيبقى y = x++, x++ من
غير brackets وهو بين أقواس كده
لو مفيش بيجري آخر فصلة على طول طيب لو بقا جوه بين أقواس بيقسم
إسناد الـ y كانت بأول فصلة وبعد (,) بقت بتاخد يعني الأخيرة هنا ه
كل الـ فيتعمل آخر الفصلة الأخيرة في الـ قبل المثال هتبقى في الأول x
وبعدين هتعمل ++ الـ x هتبقى 11 وبعدين 12 وبعدين جاي y وبعدين
تعمل ++ الـ x هتبقى 13 في الأخر خالص

x	y
~~10~~	~~10~~
~~11~~	~~11~~
~~12~~	12
13	

Date:

(32) int i=3;
printf("%d", (++i)++);

* أول ما تشوف الحركة دي اعرف انها compile error ليه لأن لما يعمل increment لل i هتكون 4 وبعدها هتعمل ++ للـ 4 دي اصلا مينفعش لأنه رقم عادي مش متخزن في الـ memory اصلا انما i increment فبيقى error

(33) int i=10;
printf("%d,%d", ++i, i++);

* قولنا قبل كده لو ماقاللك ++ في الـ printf هتمشي من اليمين للشمال دي لو محاكش compiler dependent في الاختيارات
هتبقى بكل بساطة هتروح تشوف i++ يعني هتزود الـ i وتزود عليها 1 في الـ memory
وبعد كده تدخل على الـ term التاني i++ هتزود الأول وبعدين ترجع يعني هيطبع
في الآخر 12,10

(34) int x, i=4, j=7; x=j || i++ && 7

هنا هتشوف x=j || i++ && 7 هنا مش مهم اصلا تشوف الطرف الأيسر
الـ || لأن الـ j ≠ 0 مبتاويش الصفر يعني الشرط True يعني
(1) يعني الـ x=1 والـ i=4 عادي

(35) int i=0, j=1, k=2, m;
m = i++ || j++ || k++
printf("%d%d%d%d", m, i, j, k);

* هنا في الـ m مع تطبيق ++ i هتبقى بـ 0 وبعدين يخزن +1 في الـ i وهيدخل
على || هيلاقي الـ j بـ (1) فمش هيكمل لأنه لقى True في يعني الـ m=1
والـ i زاد عليها 1 يعني بـ (1) والـ j زاد عليها (1) يعني بـ (2) والـ k
زي ما هي 2 لأنه مدخلش عليها اصلا يعني الـ output هيبقى
1122

(36) what will be the output
printf("%x", -1<<4);

* هنا -1 عايز يعملها shift 4 يعني هنشوف الـ -1 في الـ Binary
1111 1111 1111 1111 ← -1
f f f f
1111 1111 1111 0000
f f f 0

(37) write "hello word" without semicolon:

عشان تطبع "hello word" من غير ; تحطها جوا if ذي ما هتعمل عدد الأمر

وترجع True تطبع عادي

* يعني مثلاً جيت تكتب (;;) for كده

(38) syntax error —> ...

هيطلع syntax error لأن ده غلط في اللغة البرمجية نفسها

(مستطلبش .bin)

* يعني الـ logic اللي مفروض كاتبه غلط زي مثلاً أنك

logical error —> قولت 5 = 1+1 بس هو مش هيقبل الـ compiler عادي

* ومعناه إنه مش لاقي الـ Link ده بتاعه بـ ذوات

linker error —> (مستطلبش .bin) أو بجانبه بيها تعقد الـ symbole

semantic error —> هنا هي عادي بس المشكلة إن انت

بتصنف داتا تايب لـ داتا تايب تانية من غير ما تعمل casting ليها

runtime error —> هنا بيحصل crash أثناء التشغيل يعني فيه devide

by zero (error in execution) مثلاً

(39) int i = 0x10 + 010 + 10

printf ("%x", i);

هنا هتجمع Hex و Oct و Dec مع بعض

فلازم تحول الـ Dec الأول وتجمع بس خلي بالك كده في printf

هو عاوز الـ Hex فهتحول تاني من الـ Dec للـ Hex

0x10 —> Dec = 16

010 —> Dec = 8

16 + 8 + 10 = 34

34 —> Hex = 22

(40) definition —> int x; معناه إنه بيحجز مكان في الـ Memory

Initialization —> x = 1 معناه إنه بيعمل overwrite للـ corrupted data بـ value

decleration —> هو الـ definition و الـ Initialization مع بعض

زي int x = 1

(41) a = 5, b = 4; if (a == b); printf ("Equal");

* خلي بالك هنا هيطبع Equal لأن مفيش شرط بعد الـ if ده ; معناه

إن اللي بعدها مش تبعها

(42) int i = 1; if (i++ && (i == 1)) فين &&

خلي بالك إن الـ ++ بتاعة الـ i++ بعد && وبعدين أول ما تطلع false

الـ () هنا ملهاش علاقة ده بيعمل increment فبعد الـ ...

في الأخر لأنه هنا مع الأولوية ومتعدلش

false <— 1 && 0 (2 == 1)

Date: \ \

(43) int i=2, j=5, k=10; a = i>1 ? j<1 || k<1 ? 100 : 200 : 300;

بص بقى وقسمها هنحل على الشرط الأول اللي هو i>1 ودة true

فهنشتغل الي متعلق بيه j<1 || k<1 ده الحالة اللي هنا فيها علشان true

واللي بعده false فهنا نفذ الشرط ده وهنلاقيه false وهنا نفذ الـ false

بتاعه اللي هو 200