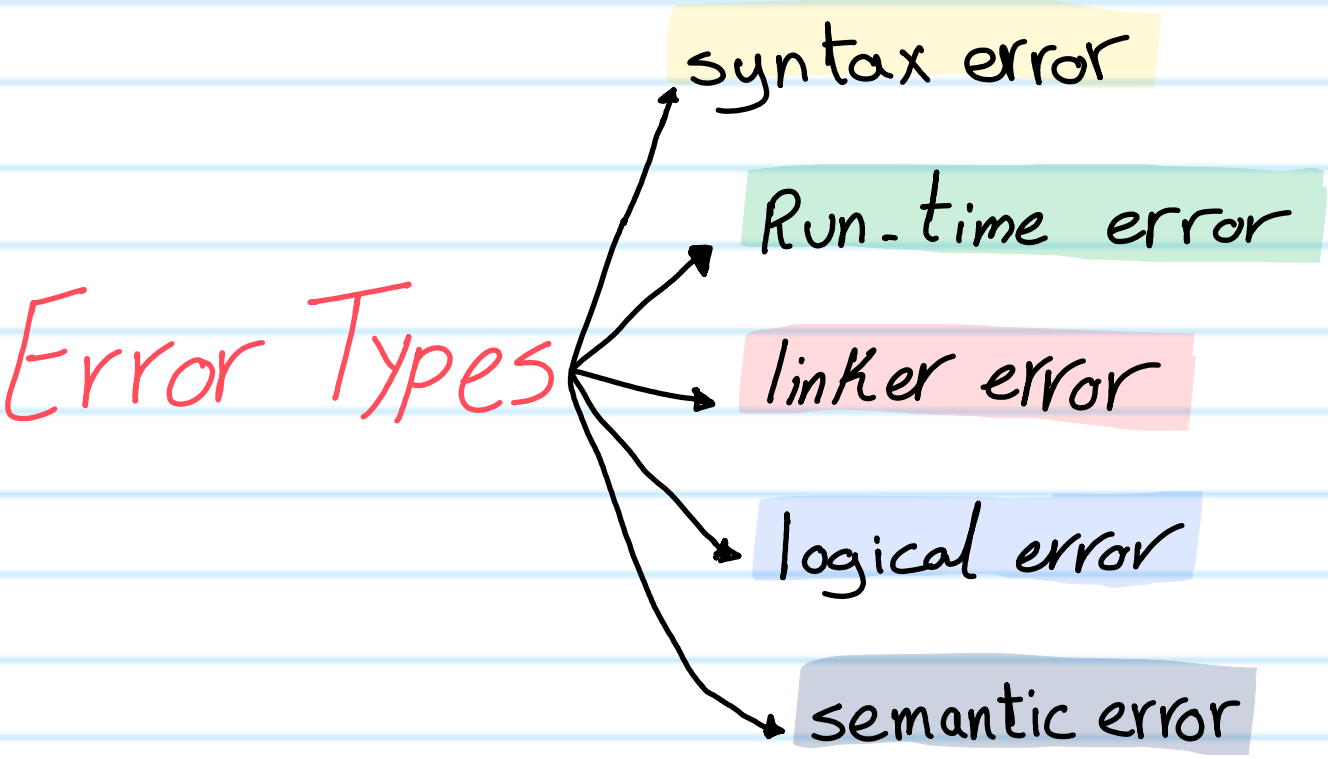
Lec 3



Programming errors are also Known as the bugs or Faults and the process of removing these bugs is Known as debugging.

debugging :: عمليه اكتشاف الخطأ (bugs) و تصحيحها عشان الكود يـ Run بشكل صحيح

1 syntax error:

- are also Known as the compilation errors as they occurred at the compilation time, or we can say that the syntax errors are thrown by the compilers.

الأخطاء دى بتحدث أثناء الكتابه ← لأنك متبعتش قواعد اللغة
الخطأ دا اللى بيعرفه (اللى بيمسك الخطأ دا) هو الـ Compiler
بيكتشفه وهو بيترجم الكود اللى أنت كاتبه و بيحوله لـ machine code
يكون فى الـ Compilation time

2 Run_time error:

- Sometimes the errors exist during the execution_time even after the successful compilation known as run_time errors.

Example:. division by Zero

```
1  #include <stdio.h>
2  int main ()
3  {
4      unsigned int Num1 = 3;
5      unsigned int Num2 = Num1/0;
6      printf("Num2 = %i \n",Num2);
7      return 0;
```

```
D:\Embedded\test\test\here.e
Process returned -1073741676 (0xC0000094)
Press any key to continue.
```

Run-time error اما تشوف حاجه زى كدا اعرف انها
returned 0 المفروض تكون

3 linker error

Linker error • Linker errors are mainly generated when the executable file of the program is not created.
• This can be happened either due to the wrong function prototyping or usage of the wrong header file.
• For example, using a variable declared without any definition.

- Linker errors are mainly generated when the executable file of the program is not created.
- This can be happened either due to the wrong function Prototyping or usage of the wrong header file.

Example: using a variable declared without any definition

```c
#include <stdio.h>
   extern unsigned int Num1;
int main ()
{
    //unsigned int Num1 = 3;
    Num1 = 3;
    printf("Num1 = %i \n",Num1);
    return 0;
}
```

declartion

definition دا كدا ال declartion وال definition تكن الكملة comments

استخدمت ال variable على طول من غير definition

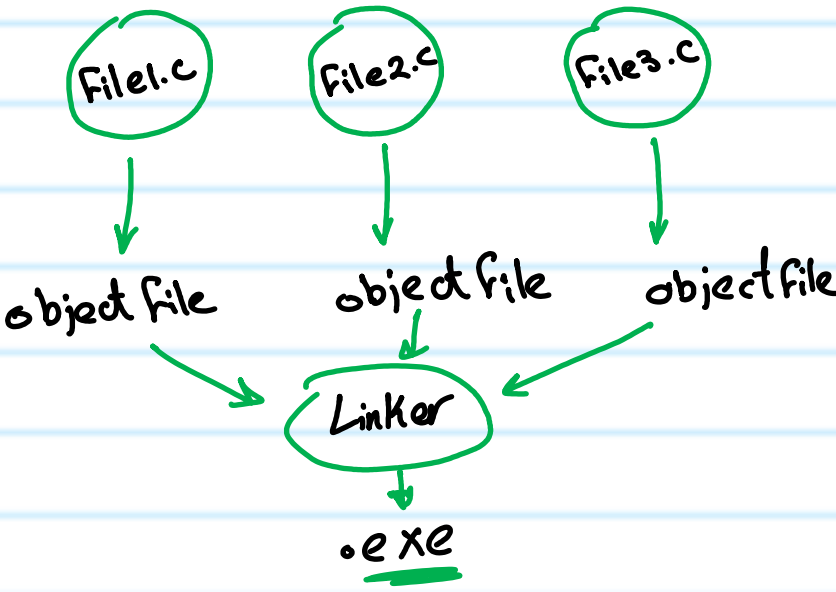
```
Build log | Build messages | CppCheck

Line  Message
      === Build file: "no target" in "no project" (compiler: unknown) ===
      undefined reference to `Num1'
      error: ld returned 1 exit status
      === Build failed: 2 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

دا شكل الخطأ

خلي الكلام دا مع ال Function بعملها declartion وراجع اعمل ال definition

ازاى ال Linker error بيحصل ؟؟

- بيكون عندى مجموعة من ال Files
- ال compiler بيتعامل مع كل File على حدا واحد ورا التانى يحول كل واحد إلى object File (machine code)
- لحد مايخلص ال Files كلها
- بعد كدا عملية ال Linker ليهم مع بعض بيحول ال object إلى .exe



4 logical error:

- The logical error is an error that leads to an undesired output.

بتكون متوقع خرج معين و تطلع حاجه تانيه

- These errors produce the incorrect output

أنت حاسب العملية والخرج مفروض كذا ويطلع رقم تاني

- mainly happens with beginners.
- The occurrence of these errors mainly depends upon the logical thinking of the developer.

5 semantic error:

statements are not understandable by the compiler

Ex:
- use uninitialized variable
- errors in expressions
- Array index out of bound
- type compatibility

المحاضرة اللي فاتت اتكلمنا على ال constant

وهي أني بخلي ال variable ← readonly مقدرش اغير قيمته

اقدر اغيره بس لو كان local بال pointers

Literals

هي طرق لكتابه ال value وليها أنواع

1 Integer Literals :

- An integer literal can be a decimal or octal or hexa decimal

الأنظمة العددية يعني

- A prefix → value بتكون قبل ال
  - ox - oX → hexadecimal
  - 0 → octal
  - nothing → decimal

- suffix is combination → value بتكون بعد ال
  - U → unsigned
  - L → long

Ex:

212 ✓

215u ✓ u → suffix means unsigned

0xFeeL ✓ ox → mean hexadecimal & L → long

078 x 0 ← قبل الرقم معناها انه octal
8 و 9 is not octal digit

032UU x خيرلسموح تكرار ال suffix من نفس النوع

prefix	suffix
85 → decimal	30 → int
0213 → octal	30u → unsigned int
0x4b → hexa decimal	30l → long
	30ul → unsigned long

مهم

① مينفعش اكتب رقم اكبر من 7 فى ال octal

② مينفعش اكرر نفس ال suffix مرتين

---

2 Floating-point literals :

- A floating-point literal has an integer part, decimal point, Fractional part and an exponent

0,0e1 ← شكل ال value بيكون كدا

- Can be decimal Form or exponential form

0,0 ↙ ↓ 0,0e1

مهم

① لازم تكتب جزء ال integer قبل العلامة العشرية مينفعش يكون فاضى

,e1 → x ,0134 ✓ 0134, → x

② لو هتكتب e لازم يتبعها رقم مينفعش بعدها فاضى

0,e → x 0,2e1 ✓

③ لو هتكتب رقم فى f لازم يكون فى decimal point

314f → x 314,1f → ✓

3 Character Literas :

single quotes ' ' بحطه بين character عشان اخزن فى اد ك
بيكون ال variable دا نوعه char

Ex: char x = 'c' ;

4 string Literas :

double quotes " " بحطه بين string عشان اخزن فى اد ك
* وبيتخزن فى Array من نوع char

Ex: char arr1[20] = " Embedded Diploma" ;

Operators

is a symbol that tells the compiler to perform specific mathematical or logical Function

Types of operators:

a) Arithmetic Operators
+ - * / % ++ --

b) Relational operators
== != > < >= <=

c) logical operators
&& || !

d) Bit wise operators
& | ^ ~ << >>

e) Assignment operators
= += -= *= /= %= <<= >>=

f) conditional operators

g) size of operators
size of( )

① Arithmetic Operators

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

أهم حاجه هنا ++ و --

++ → increment operator, increase the integer value by one

-- → decrement operator, decrease the integer value by one

++var

pre_increment

→ is used to increment the value of a variable before using it in a expression

→ in the pre_increment, value is First incremented and then used inside expression

var++

post_increment

→ is used to increment the value of a variable after executing expression completely in which post increment is used.

→ in the post_increment, value is First used in exp and then incremented

Notes

- IF we assign the post_incremented value to the same variable → the value will remain the same like it was before.

Post incremented
Post decremented

→ الكلام دا ينطبق

```c
#include <stdio.h>
unsigned int Num1 = 1;
unsigned int Num2 = 1;
unsigned int Result ;
int main ()
{
    printf("Num1 = %i \n",Num1);
    Num1++;
    printf("Num1 = %i \n",Num1);

    printf("Num2 = %i \n",Num2);
    ++Num2;
    printf("Num2 = %i \n",Num2);

    Num1--;
    printf("Num1 = %i \n",Num1);

    --Num2;
    printf("Num2 = %i \n",Num2);


    return 0;
}
```

```
D:\Embedded\test\test\here.e
Num1 = 1
Num1 = 2
Num2 = 1
Num2 = 2
Num1 = 1
Num2 = 1

Process returned 0 (0x0)   executic
Press any key to continue.
```

تأثير الـ pre والـ post مش هيظهر هنا عادي
ولكنه هيظهر بمجرد استخدامه في معادله

هنا واضح التأثير

here.c

```c
#include <stdio.h>
signed int Num1 = 1;
signed int Num2 = 1;
signed int Result ;
int main ()
{
    printf("Result = %i \n",Result);
    Result = Num1 + Num2;
    printf("Result = %i \n",Result);
    Result = 0 ;
    printf("Num1 = %i \n",Num1);
    Result = ++Num1 ;
    printf("Num1 = %i \n",Num1);
    printf("Result = %i \n",Result);
    Result = Num1++ ;
    printf("Num1 = %i \n",Num1);
    printf("Result = %i \n",Result);
    Result = 0;
    printf("Num2 = %i \n",Num2);
    Result = --Num2;
    printf("Num2 = %i \n",Num2);
    printf("Result = %i \n",Result);
    Result = Num2--;
    printf("Num2 = %i \n",Num2);
    printf("Result = %i \n",Result);
    return 0;
```

```
D:\Embedded\test\test\here.e
Result = 0
Result = 2
Num1 = 1
Num1 = 2
Result = 2
Num1 = 3
Result = 2
Num2 = 1
Num2 = 0
Result = 0
Num2 = -1
Result = 0

Process returned 0 (0x0)   execution time : 1.142 s
Press any key to continue.
```

pre
post
pre
post

ere × here.c ×

```c
#include <stdio.h>
signed int Num1 = 1;
signed int Num2 = 1;
signed int Num3 = 2;
signed int Num4 = 2;
int main ()
{
    printf("Num1 = %i \n", Num1);
    printf("Num1 = %i \n", ++Num1);
    printf("Num1 = %i \n", Num1++);
    printf("Num1 = %i \n\n", Num1);

    printf("Num2 = %i \n", Num2);
    printf("Num2 = %i \n", Num2++);
    printf("Num2 = %i \n", Num2);
    printf("Num2 = %i \n", ++Num2);
    printf("Num2 = %i \n\n", Num2);

    printf("Num3 = %i \n", Num3);
    Num3++;
    printf("Num3 = %i \n", Num3);
    Num3 = Num3++;
    printf("Num3 = %i \n\n", Num3);

    printf("Num4 = %i \n", Num4);
    ++Num4;
    printf("Num4 = %i \n", Num4);
    Num4 = ++Num4;
    printf("Num4 = %i \n\n", Num4);
```

D:\Embedded\test\test\here.e ×

```
Num1 = 1
Num1 = 2
Num1 = 2
Num1 = 3

Num2 = 1
Num2 = 1
Num2 = 2
Num2 = 3
Num2 = 3

Num3 = 2
Num3 = 3
Num3 = 3

Num4 = 2
Num4 = 3
Num4 = 4
```

② Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

بتدرس علاقه حاجه بحاجه

check operator ممكن نسميه

بيكون عبارة عن سؤال هل قيمة الـ A [علاقة] قيمة B ؟

true = 1

Flase = 0

الاجابه بتكون

ناتج العمليه بيكون Flase or true

بستعملها أكتر مع if condtion & For

here.c

```c
#include <stdio.h>
signed int Num1 = 1;
signed int Num2 = 1;

int main ()
{
    printf("Result = %i \n", (Num1 == Num2));
    printf("Result = %i \n", (Num1 != Num2));
    printf("Result = %i \n", (Num1 > Num2));
    printf("Result = %i \n", (Num1 < Num2));
    printf("Result = %i \n", (Num1 >= Num2));
    printf("Result = %i \n", (Num1 <= Num2));
```

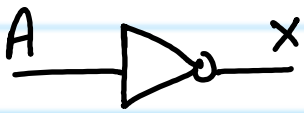
```
D:\Embedded\test
Result = 1
Result = 0
Result = 0
Result = 0
Result = 1
Result = 1
```

③ Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

ناتج العمليه دى برضو 1 أو 0

فكرتها جايه من ال Logic gates

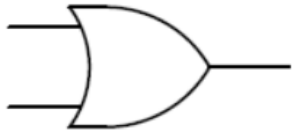


not

A	x
0	1
1	0

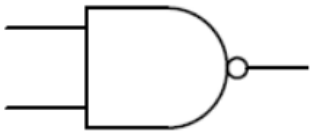
AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



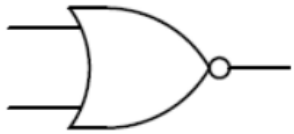
OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



NOR

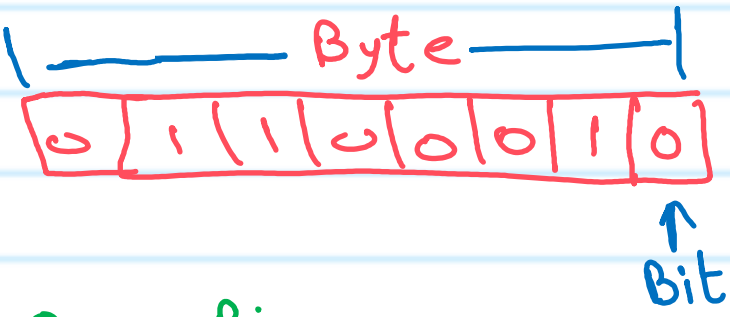
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

دا مهم لتأثير ال Logical operators

Bit and Bytes

1 Byte = 8 bit

bit ⇄ 0 1

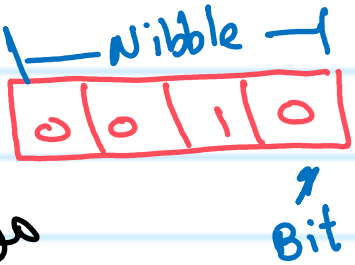


decimal → 98 = 01100010 ← Binary

$98 = 2^6 + 2^5 + 2^1$

lowest value 0 0 0 0 0 0 0 0 = 0

Highest value 1 1 1 1 1 1 1 1 = 255



Nibble = 4 bit

موجود فى الـ MCU القديم

---

* Word الـ Sizes الـ word بيعتمد على الـ Architecture

1 word ← 16 bit ← 2 byte على الأقل بتكون

2 word ← 32 bit ← 4 byte لو حبتى

DWORD doubleword بيقولوا عليها

8 word ← 64 bit ← 8 byte لو حبتى

LWORD long word بيقولوا عليها

here.c

```c
#include <stdio.h>
signed int Num1 = 1;
signed int Num2 = 0;

int main ()
{
    printf("Result = %i \n", (Num1 && Num2));
    printf("Result = %i \n", (Num1 || Num2));
    printf("Result = %i \n", (!Num1));
    printf("Result = %i \n", (!Num2));
    printf("Result = %i \n", (!Num1 && Num2));
    printf("Result = %i \n", (!Num1 || Num2));
    printf("Result = %i \n", (!Num1 || !Num2));


    return 0;
}
```

```
D:\Embedded\test\test\here.c
Result = 0
Result = 1
Result = 0
Result = 1
Result = 0
Result = 0
Result = 1

Process returned 0 (0x0)   execu
Press any key to continue.
```

(4) Bitwise Operators

Operator	Description	Example A=60 B=13
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

الـ value هنا مش true or false (1,0)

هنا العمليه بتتم على الـ Bit و بيطلع ناتج (بيحول قيمة A ، B إلى Binary

و يتم العملية لكل bit مع اللي قصدها

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows –

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

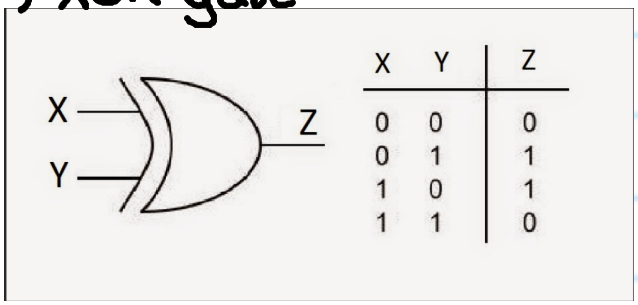
A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

شبه بعض ← 0
مختلفين ← 1

(^) XOR gate



الـ Bit wise operator بستخدمها مع الـ bit manipulation عمليات على الـ bit

مثلاً" معايا متغير وعاوز الغير Bit هيئه اخليها بـ صفر بعمل AND لـ الـ bit دى بصفر

Ex: unsigned char Btn_state= 0x55; /* 0101 0101 */

عاوز لكل bit هيئه اخليها بصفر أو واحد

هعمل حاجه اسمها bit mask ← دى عباره عن متغير تانى من نوع unsigned char

unsigned char Btn_state_1= 0b.....

هشوف أنا عاوز احط انه bit بصفر واحطها فى الـ mask بصفر

واللى مش عاوز اغيره احط فيها 1 و بعدكدا اعمل ليهم AND (&) سوا

Btn_state= Btn_state & Btn_state_1

عاوز الناتج يكون 0000 0101 ← فالـ mask يكون كدا 0b 1111 0000 واعمل AND (&)

عاوز الناتج يكون 1111 0101 ← فالـ mask يكون كدا 0b 0000 1111 واعمل OR (|)

وهكذا اشوف اللى عاوز اعمله واختار العمليه اللى تناسبه

Bitwise AND الـ : &

بستعملها لو عاوز اخلى bit بـ صفر

clear

Bitwise OR الـ |

بستعملها لو عاوز اخلى bit بـ 1

set

Bitwise XOR الـ ^

بستعملها فى عملية الـ toggle

Bitwise Not الـ ~

بتعكس الـ bit الـ 0 ← 1 ، الـ 1 ← 0

1's Complement

Bitwise AND (&)

Bitwise OR (|)