Lec 4

*integer promotions

اما باجى اعمل اى operation لأى variable الـ data type بتاعه من نوع short int / char

تلقائى بيحصل integer promotions عشان يتفادى الـ over flow فى عملية التنفيذ فقط

لكن فى النتيجة بيرجع كل حاجه لأصلها

المتغيرات من نوع char

فى الحته دى حصل integer promotions

```
1   #include <stdio.h>
2   char Num1=30;
3   char Num2=40;
4   char Num3=10;
5   int main()
6   {
7       char Result = (Num1*Num2)/Num3;
8       printf("Result = %i \n", Result);
9       return 0;
10  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\Embedded\VsCodeProjectTemplate> .\EmbeddedDiploma.exe
Result = 120
PS D:\Embedded\VsCodeProjectTemplate>
```

30 X 40 = 1200

الـ char (-128 : 127)

فاللحظة دى حصل الـ integer promotions

لأن الـ variable من نوع char

وفى اثناء العملية بيترقى ويبقى integer عشان الـ over flow

ودا الناتج و اتخزن عادى $\frac{1200}{10} = 120$

فالـ char محصلش over flow

unsigned char

أنا مخزن نفس الرقم هو فى char ومرفض

لو طبعتهم كـ char هيطلع ليك نفس الشكل ✓ ومع ذلك No

ودا لأن فى عملية المقارنة ==

الـ integer promotions حصل لـ 2 variable

فى اللحظة دى قيمتهم كـ integer

num2 = 251 < num1 = -5 الـ

فعشان كدا الناتج NO

```
1   #include <stdio.h>
2   char num1= 0xFB;          // 251 → -5
3   unsigned char num2= 0xFB; // 251
4   int main()
5   {
6       printf("Num1 = %i \n",num1);
7       printf("Num2 = %i \n",num2);
8       if (num1 == num2){
9           printf("yes \n");
10      }
11      else{
12          printf("No \n");
13      }
14      return 0;
15  }
```

هنا المشكله

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\Embedded\VsCodeProjectTemplate> .\Embed
Num1 = -5
Num2 = 251
No
PS D:\Embedded\VsCodeProjectTemplate>
```

كل الحوار هو لما ييجى تعمل operation

لمتغيرات short < char

بتتحول من الوقت دا لـ int

القيمة بتتحول لـ binary و يشوفوا

تتخفض تتخزن و تفضل نفس القيمة والـ type يتغير

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

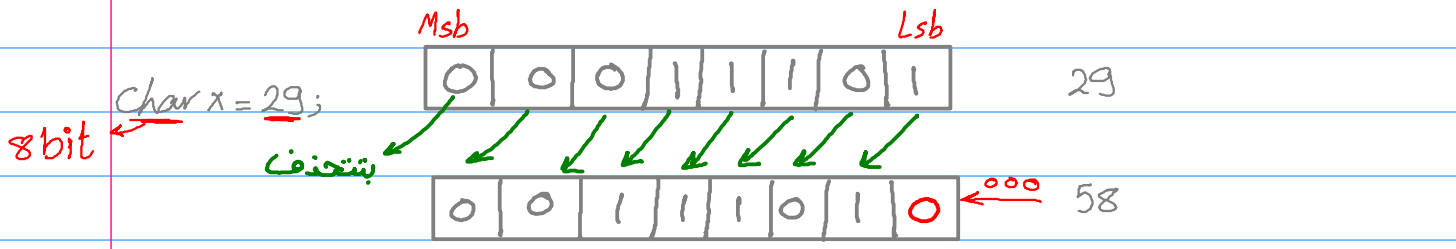
Binary shift

- Logical shift
- Arithmetic shift

Logical shift

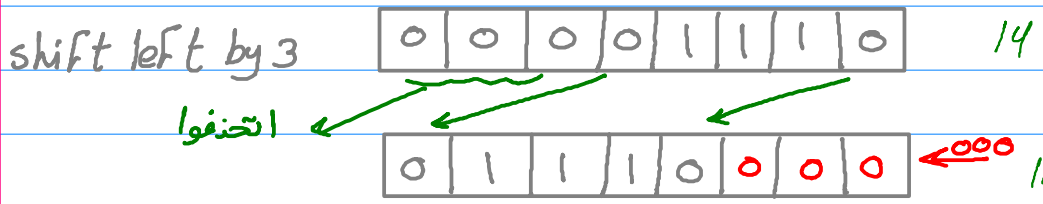
1- logical shift left :

→ shift left operation shifts each bit one place to left



عملت هنا shift Left وكنا بالشكل دا ↑ ازاحه لجمه اليسار لجميع الـ bits واضاف صفر من اليمين

- In Binary, left shifting multiplies by 2 , because we are working in base 2



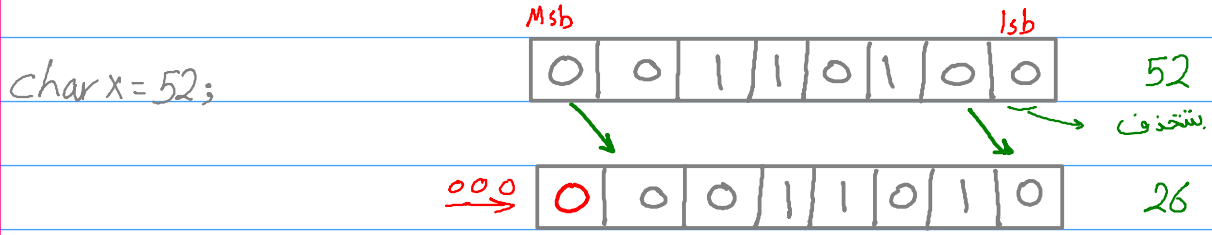
14 X2 → X2 → X2 = 112

كل shift left اضرب 2 X

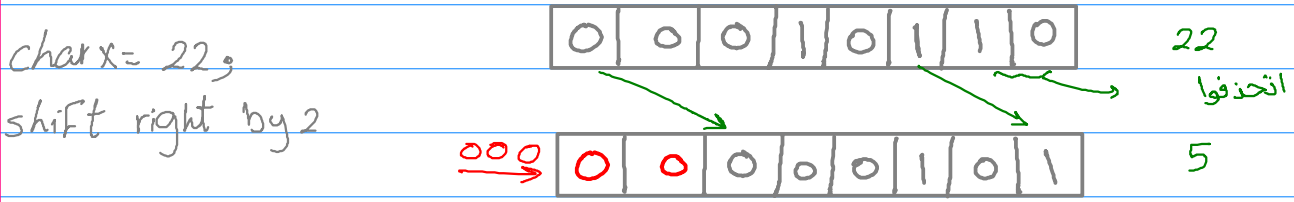
Logical shift

2_logical shift right :

→ shift right operation shifts each bit one place to right



- In Binary, right shifting divides by 2 , because we are working in base 2



22÷2 → ÷2 = 5

5,5 الكسر مش بيتكتب

الملخص

- في عمليه الـ shifting لازم اخلي بالي من عدد الـ bits 8، 16
لأن العمليه دي هتأثر على قيمة المتغير اللي بعمله shift

- بستخدم الـ bitwise shift ككود shift عشان انفذ عمليه الـ >> ، <<

- كدا بعمل Bitwise shift left لـ 29 • 29<<1
١- هتتحول لـ Binary ٢- بتعملها shift left مره واحده

- في عمليه الـ shift left الرقم اللي بعمله كدا بيضرب 2x لو مره واحده
لو بعمل اكتر من مره هضرب x في $2^{\square}$ أس عدد الـ shift

29<<2 → $29 \times 2^2 = 116$

- لو هعمل shift لعدد فردي بهمل الـ 0,5 والناتج بيكون عدد صحيح

الـ Binary ← الـ base=2 عشان كدا
Left → بضرب 2x
Right → بقسم 2÷

ملاحظه • الاهمال بيكون فى اتجاه الاصغر

1,5 → 1

-1,5 → -2

• لو عملت shift لرقم فردى وقولتله يظهر الرقم كـ Float برضو هيظهر رقم صحيح من غير اى 0,5 لأن فى الميموري من الاساس الـ bit مش بتخزن النص

```c
#include <stdio.h>
unsigned int num1 = 5;
unsigned int num2 = 52;
int main()
{
    printf("Result = %i \n", (num1 << 1)); // << shift left *2
    printf("Result = %i \n", (num1 << 2)); // *2 *2
    printf("Result = %i \n", (num1 << 3)); // *2 *2 *2

    printf("Result = %0.3f \n",(float) (num2 >> 1)); // >> shift right /2 (52/2 = 26) 26.000
    printf("Result = %0.3f \n",(float) (num2 >> 2)); // (26/2 = 13) 13.000
    printf("Result = %0.3f \n",(float) (num2 >> 3)); // (13/2 = 6.5) 6.000
    printf("Result = %0.3f \n",(float) (num2 >> 4));  // (6.5/2 = 3.25) 3.000
    printf("Result = %0.3f \n",(float) (num2 >> 5)); // (3.25/2 = 1.625) 1.000
    return 0;
}
```

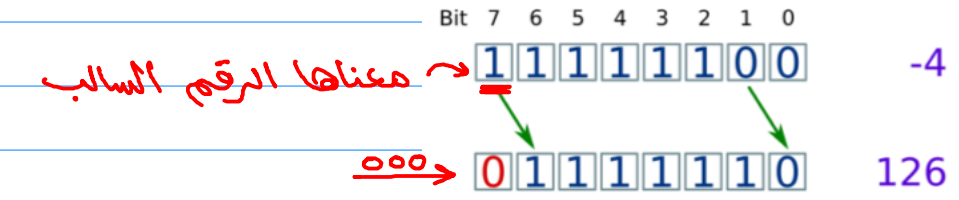
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\Embedded\VsCodeProjectTemplate> .\EmbeddedDiploma.exe
Result = 10
Result = 20
Result = 40
Result = 26.000
Result = 13.000
Result = 6.000
Result = 3.000
Result = 1.000
```

• If you are dealing with bytes or words which represent signed integers, the logical shift right won't work For negative numbers

0 → positive

1 → negative



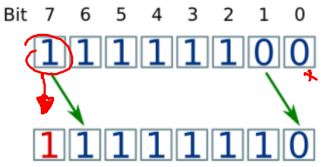
Arithmetic shift عشان كدا هنلجأ لـ unlogical طبعا العمليه دى

• Arithmetic shift

→ the sign bit stays the same as the data shifts.

* IF the sign bit was 0, it will be Filled with a 0, but if it was 1 it will be Filled with a 1

اخر bit تنزل زى ما هى
وتتعمل بعدد الـ shift للرقم عادى



shift right ÷2

$\frac{-4}{2} = -2$ ✓

1- Arithmetic shift right

$\frac{-3}{2} = -1.5 \rightarrow$ round -2 round للصغير

-3 ← 1111 1101
-2 ← 1111 1110

2- Arithmetic shift left

مش محتاجه بث logical shift left بيعمل نفس الشئ
بعمل shift لليسار واحط صفر من اليمين

فى المحاضرة اللى فاتت بدأنا فى الـ Bit wise operator
وأنى كنت بستخدمه فى الـ bit manipulation

1 لو أنا عايز اعمل clear لـ bit معينه (فى معادله جاهزه استخدمها هتوفر عليك كتير)

```
Variable &= ~(1 << Bit_position);
```

```
Variable = variable & ~(1 << Bit_position);
```

المتغير اللى عاوز اعمله clear ← AND → ~ Bitwise Not ← رقم واحد 0000 0001

← الـ bit اللى عاوز اعملها clear

Ex: 1001 0011 (7 6 5 4 3 2 1 0)

عاوز اعمل لها clear

Variable = 1001 0011 & ~(1<<4)
= 1001 0011 & ~(0001 0000)
= 1001 0011 & (1110 1111) = 1000 0011

0000 0001 <<4
0001 0000
~ 1110 1111

1001 0011
1110 1111 &
1000 0011

الـ bit المطلوب بقت بصفر

2 لو أنا عايز اعمل set لـ bit معينه (استخدم المعادله دى)

```
variable |= (1<< Bit_position);
```

```
variable = variable | (1<< Bit_position);
```

المتغير اللى هيتم عليه الـ set | OR | 0000 0001 | الـ bit اللى عاوز اعملها set

Ex: variable = 1 0 0 1 0 0 1 1
7 6 5 4 3 2 1 0

عاوز اعمل الـ set لـ bit دى

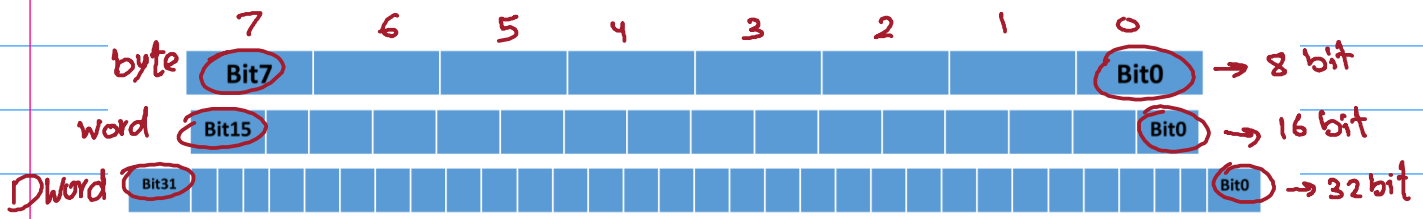
```
variable = variable | (1<< Bit_position);
         = 10010011 | (1 << 5);
         = 10010011 | 0010 0000
         = 10110011
```

0000 0001 << 5 = 0010 0000

```
10010011
00100000
--------
10110011
```

الـ bit اللى عاوزها اتعمل لها set

هم نبقا عارف رقم الـ bit عشان نقدر نستخدم المعادلتين دول



ابدأ عد من اليمين من رقم صفر

---

3 لو أنا عايز اعمل toggle لـ bit معينه (استخدم المعادله دى)

(يعنى لو هى بـ 1 نبقا صفر ، لو هى صفر نبقا 1) هعكس الـ bit يعنى

```
variable ^= (1 << position);
```

```
variable = variable ^ (1<< Bit_position);
```

المتغير اللى هيتم عليه العمليه | XOR | 00000001 | الـ bit اللى عاوز اعملها toggle

XOR
لو متشابهين بصفر
لو مختلفين بـ 1

Ex: variable = 10010011
7 6 5 4 3 2 1 0

حاوز اعمل لها Toggle

Variable = variable ^ (1<<Bit_position)

= 10010011 ^ (1<< 5)

= 10010011 ^ 00100000

= 10110011

"وعدلا" Toggle اعمل ليها

10010011
00100000^
10110011

0000 0001<<5

0010 0000

⑤ Assignment Operators

The following table lists the assignment operators supported by the C language –

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Assignment ← يعني اضف اللي في اليمين احطه في الشمال

ابسط مثال : number = 5;

& ← لو اتحط قدام اي variable هيجبلك عنوانه (address) في الـ memory

* ← هتتشوف مع الـ pointer (لو اتحطت قبل عنوان هتجبلك القيمة اللي في العنوان دا)

scanF Function

& ← بستخدمها مع

• دالة بستخدمها عشان استقبل فيها data من الـ user

```
scanF ("%i \n" , &a);
```

↳ اى specifier عمومًا

↳ الـ variable اللى هستقبل فيها الـ data

لازم يكون قبله علامة &

عشان القيمة توصل لعنوان الـ variable

```
scanF ("%i %d %X \n", &a &d &c);
```

↳ استقبال data فى اكتر من variable

Conditional Operator (?:)

is a ternary operator and it takes three operands

```
variable = Expression1 ? Expression2 : Expression3
```

condition

* فى المعادلة هنا Expression1 هنعتبر مكانها شرط

• لو الشرط دا اتحقق هينفذ اللى موجود فى Expression2

• لو متحققش هينفذ اللى فى Expression3

بمعنى بتسلك سلوك الـ (iF) لو مفيش variable

• لو فى variable فى الـ Expressions دى هتم والناتج منها هيتخزن فى الـ variable

---

Operator precedence

اساس الـ online Exams

1- decides how an expression is evaluated

2- certain operators have higher precedence than others.

* ترتيب العمليات فى التنفيذ (الأولويات)

يعنى لو معادلة فيها ضرب وجمع مين هينفذ الأول؟ الـ precedence هو اللى بيحدد

Associativity

1- is used when two operators of same precedence apper in expression
2- can be either left to right or right to left

---

⑥ precedence Operator

Category	Operator = precedence	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

اللى موجود الأول له الأولوية

يعنى ال * / % لهم أولوية عن + -

لأنهم موجودين فى الجدول الأول

ملحوظة: اى عمليه تتم داخل اد sizeof( ) لاتتغير فى قيمة الoperand

Decision Making

* if statement
* if else statement
* if elseif....else statement

هناخد نظره سريعه عنهم و نكمل المحاضره الجاية

① if statement

```
if ( condition ..... )
{
   statement1;
   statement2;
      ⋮
}
```

ممكن شرط واحد
او عده شروط

← الـ if statement طريقة عملها كالآتى

- لو الـ condition اتحقق يعنى بقا True هينفذ العمليات اللى البلوك اللى بعده { }
- لو الـ ~ متحققش يعنى بقا False مش هيدخل على { } و هينزل ينفذ باقى الكود ايا كان اى

* لو الـ condition اتحقق و مكنش فى { } هينفذ أول جمله بعد الـ if / الباقى هيتم تنفيذه لكن كـ كود عادى

② if...else statement

```
if ( conditions ..... )
{
   statements;
      ⋮
}
else
{
}
```

← لو الـ conditions متحققش
هينزل ينفذ فى الـ else على طول
← لو الـ conditions اتحقق.. هينفذ اللى { } تحت الـ if
و هيهمل باقى الكود

و بستخدم دى علشان أوفر وقت فى الـ compile

③ else if statement

```
if ( conditions ①)
{ statements;①
  ⋮
}
else if ( conditions②)
{ statements; ②
        ⋮
}
⋮
else
{
}
```

else if

- أول بشوف شرط الـ if لو اتحقق
  فهينفذ الـ block اللي بعده و هيهمل باقي الكود اللي هو else if / else
- لو متحققش شرط الـ if هينزل يشوف الشرط اللي بعده
  ثم اللي بعده لحد ما شرط يتحقق و ينفذ الـ block بتاعه
- لو مفيش اي شرط اتحقق هينزل ينفذ اللي في الـ else

3/10/23