lesson 39 Bitwise operator NOT

سنتعلم فى هذا الدرس علامة **NOT** و التي وظيفتها تغيير العدد ال binary بحيث تجعل كل 0 يتحول إلى 1 ، وكل 1 يتحول إلى 0 .
علامة ال ~ لا تحتاج رقمين لتقوم بعمل عملية ما بينهم، هي فقط تغيير الرقم الذى وُضع أمامها :

```
int x = 5;
printf( " %d ", ~x );
```

output :
-6

```
int x = -5;
printf( " %d ", ~x );
```

output :
4

سنكتشف أن ال not ~ تقوم بتحويل الرقم الموجب إلى رقم سالب أكبر منه ب1، و تقوم بتحويل الرقم السالب إلى رقم موجب أقل منه ب 1.

قد يظهر الأمر انه بسيط لكنه معقد بعض الشئ، لأن أى رقم من نوع int لديه 32 **bits**، وهذا معناه ان رقم 5 مثلاً ال **binary** الخاص به هو :

00000000000000000000000000000101

عند استخدام **not** (~) سيتحول كل الارقام التى تساوى 0 الى 1
وفى رقم 5 مثلاً سيتحول كل الاصفار التى على اليسار إلى 1

11111111111111111111111111111010

و في ال bits أخر رقم على اليسار هو ما يحدد إذا كان الرقم موجب أم سالب، فإذا كان 0 هذا معناه أن الرقم موجب و إذا كان 1 هذا معناه أن الرقم سالب
لذلك كان ~5 = -6 ، و لكن كيف يحدث ذلك ؟

لنفترض أن عدد ال bits الخاصة بال int هو 4 bits فقط (هذا كلام افتراضى و ليس الواقع)

فى هذه الحالة ستكون الأرقام المتاحة للتخزين في هذا ال int هي :

decimal	binary
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
1-	1001
2-	1010
3-	1011
4-	1100
5-	1101
6-	1110
7-	1111

حيث أننا أفترضنا أن ال int له 4 bits فقط ، في هذه الحالة الرقم الذى يمكن التسجيل فيه سيكون مكون من 3 خانات binary و هم ال3 خانات على اليمين، و الخانة الأخيرة التي

في أقصى اليسار هي التي تحدد الإشارة، إذا كانت 0 معناها الرقم موجب و إذا كانت 1 معناها الرقم سالب

و لكن إذا قمنا بالتركيز على الترتيب الذى حصل للرقم ال binary في الجدول السابق ، هناك رقم غبر موجود و هو عندما وصل الجدول إلى ال 0 و كان ال binary الخاص به هو 0000 كان يجب أن يكون الرقم التالى له هو 1000 و لكن في هذه الحالة كان سيكون رقم بلا معنى لأنه سيعبر عن (0 -) و هو رقم ليس له وجود.

و الشئ الأخر أيضا أنه لا يوجد أمر مباشر لتحويل الخانة الأخيرة في الرقم ال binary فقط دون تحويل باقى الخانات.

لذلك لحل هذه المشاكل تم الاتفاق على التالى :

أولا : لتحويل رقم من موجب إلى سالب، لن يتم تحويل الخانة الأخيرة في ال bits فقط، بل سيتم تحويل كل الخانات من 0 إلى 1 و العكس، و في هذه الحالة الأمر المباشر الذى يمكننا من ذلك هو not (~)

ثانيا : ال (~ 0) يساوى -1 لأن رقم 0- ليس له وجود و إذا حجزنا له مساحة فارغة، هذا سيكون هدر للذاكرة

في هذه الحالة الجدول سيكون كالتالى :

decimal	binary		decimal	binary
7	0111		-8	1000
6	0110		-7	1001
5	0101		-6	1010
4	0100	=	-5	1011
3	0011		-4	1100
2	0010		-3	1101
1	0001		-2	1110
0	0000		-1	1111

~

في هذه الحالة، الخانة التي تم حذفها (0-) تم الإستعاضة عنها ب 1- لذلك ال ~0 = -1
و ذلك أدى إلى عمل الفارق الذى يجعل عند استعمال علامة ال not ~ في الكود سنكتشف أن الناتج دائما يكون زائد رقم أو ناقص رقم مع تغيير الإشارة .

يمكنك ملاحظة أن عندما إفترضنا أن ال int له مساحة 4 bits فقط، كان أكبر رقم موجب يمكن تسجيله فيه هو رقم 7 و كان يقابله رقم 8- عند عمل عملية ال ~ له .
و رقم 8- في ال binary كان عبارة عن 1 في أقصى اليسار و على يمينه مجموعة من الأصفار.
وهنا يمكننا استخدام هذا المبدأ لإكتشاف أكبر رقم يمكن تسجيله في أي مجموعة من ال bits عن طريق وضع 1 في أقصى يسار هذه المجموعة ثم تحويل الرقم الناتج باستخدام عملية ال not ~

مثال لاكتشاف اكبر رقم موجب يمكن تخزينه في ال int :

```
int main(  ) {
  int x = 1<<31;
  //هنا قمنا بعمل  left shift  لل 1 لنحصل على أكبر رقم سالب
  int z = ~x;
  // سيقوم بتحويل الرقم السالب الى المناظر له الموجب
  printf("%d", z);
}
```

output:

2147483647

(قم بتجربة الكود بنفسك واضغط هنا)