

## Data Representation:

As mentioned before, a computer can only store binary numbers. Therefore, a character, a Boolean etc. are actually represented by binary numbers. For example, a computer stores ASCII code to represent a character, it uses 0 to represent FALSE and values other than zero to represent TRUE.

So, it is possible to change the type of a variable, like from Char to integer. This is called **type casting**.

In Pascal, we use the following syntax to change a type:

<i>Datatype</i> (Value)
-------------------------

### Example:

begin writeln(integer('a')); end.	begin writeln(integer(false)); end.	begin writeln(char(97)); end.	begin writeln(byte('A')); end.
97	0	a	65

'a' is represented as 97, which is 01100001 (ii) in binary representation. So, changing 01100001(ii) into an integer and output it will get 97. False is represented as 0, casting it to integer will get zero.

*Notes: computers always group 8 binary digits, which are called bits, to form 1 byte to perform calculation.*

### Weird Output

begin writeln(integer(65535)); end.	begin writeln(longint(integer(2147483647))); end.	begin writeln(integer(65536)); end.
-1	-1	0

  

Var c:char; Begin Write(pred(Boolean(2))); end.	Var i:integer; Begin I:=-32768; write(-i); end.	Var i:integer; Begin I:=32767; write(2*i); end.
TRUE	-32768	-2

*Be careful when casting between data types not having same sizes.*

## Base (進制)

We use denary representation in our daily life, but there are other bases, like binary, octal, hexadecimal.

How can we change the number in one form to another form?

In Primary School, you have already learnt how to change a denary number to binary.

There are two ways to do so.

1. By short-division, which has been taught in Primary School.
2. Using Addition.

Let's see how to use the second method.

Convert 234 (x) to binary.

Find a power of two just less than the number.

( $2^8 = 256$ ,  $2^7 = 128$ ) (So, we choose 128)

Then subtract 128 from the number, and write a **1** on 8<sup>th</sup> digit.

Afterwards, the number becomes 106.

We do this process recursively.

Find a power of two just less than the number.

$2^6 = 64$

Subtract 64 from the number, and write a **1** on 7<sup>th</sup> digit.

It becomes 42

$2^5 = 32$

Subtract 32 from the number, and write a **1** on 6<sup>th</sup> digit.

It becomes 10

$2^4 = 16$ , bigger than 10, so we write a **0** on 5<sup>th</sup> digit.

$2^3 = 8$ , write a **1** on 4<sup>th</sup> digit, and subtract it.

$2^2 = 4$ , a **0** is written on 3<sup>rd</sup> digit.

$2^1 = 2$ , a **1** is written on 2<sup>nd</sup> digit.

$2^0 = 1$ , a **0** is written on 1<sup>st</sup> digit.

2		234	
2		117	... 0
2		58	... 1
2		29	... 0
2		14	... 1
2		7	... 0
2		3	... 1
2		1	... 1
		0	... 1

So

$234(x) = 11101010(ii)$

If we want to change a number from binary to octal or hexadecimal, should we change it to denary and use short-division to find out binary?

No.

We'd better use another method instead. (**Bit-Partitioning**)

Let's see the table of Dec, Bin, Oct and Hex.

Denary	Binary	Octal	Hexadecimal
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

E.G.

11101010 (ii)

Bin→Oct

We break it into groups with 3 digits each from right hand side.

i.e. 011 101 010

Look at the table above and change those 3-digit binary numbers into Octal one.

011 = 3 (viii)

101 = 5 (vii)

010 = 2 (viii)

Therefore, the octal representation of 234(x) is 352(viii).

Bin→Hex

We break it into groups with 4 digits each from right hand side.

i.e. 1110 1010

Look at the table above and change those 4-digit binary numbers into Hex one.

1110=E (xvi)

1010=A (xvi)

Therefore, the hexadecimal representation of 234(x) is EA(xvi).

## Bitwise Operator:

Please first refer to all ppt and pdf files in Lesson12 for 2's complement representation.

### Not, And, Or, Xor.

You have learnt about these four operators for Boolean expressions, i.e. all operands are Boolean type.

This time, we will learn about when all the operands are integer-type.

In this case, such operations are called bitwise operation.

Namely, we consider the bits of the operands instead of its truth value.

For the **Not** operator, it will turn all 0's to 1's and 1's to 0's of a binary value.

Example: Calculate "Not 10"

10 = 00001010 (ii)

```
Writeln(not 10);
```

```
Writeln(byte(not 10));
```

Not 10 becomes 11110101 (ii).

It represents a signed-integer -11 or an unsigned-integer 245.

**And** is liked the logical **And**, Only when two bits are 1, it gives 1.

So, "1 and 2" is equal to 0. "1 and 3" is equal to 1.

Explanation:

1 = 01 (ii)

2 = 10 (ii) AND

00 (ii) = 0

**Or** is also similar to logical **Or**, it gives 0 only if two bits are 0.

So, this time "1 or 2" is equal to 3. "1 or 3" is equal to 3.

Explanation:

1 = 01 (ii)

2 = 10 (ii) OR

11 (ii) = 3

Finally, the **XOR** operator, which is the most powerful bitwise operator, it gives 1 only if either bit is 1.

So, "1 xor 3" = 2. "1 xor 2" is equal to 3.

Code for swapping a and b:

1 = 01 (ii)

3 = 11 (ii) XOR

10 (ii) = 2

```
A:=a xor b;
```

```
B:=a xor b;
```

```
A:=a xor b;
```

### The shift operator: shl , shr.

Shl is shift left, while Shr is shift right.

"A shl b" means shift the bit pattern of A to left by adding b's zero at the rightmost, and drop the leftmost bits if they exceed the size of its type.

So, for a byte, which can store 8-bit only. Example: "byte(128 shl 3)" will get zero.

Explain: 128 = 10000000(ii) shift left by 3 → 10000000000, the leftmost 3 bits exceed 8 bits, so drop them and get 00000000(ii) = 0.

Shr is similar to shl, it shifts a bit pattern to right.