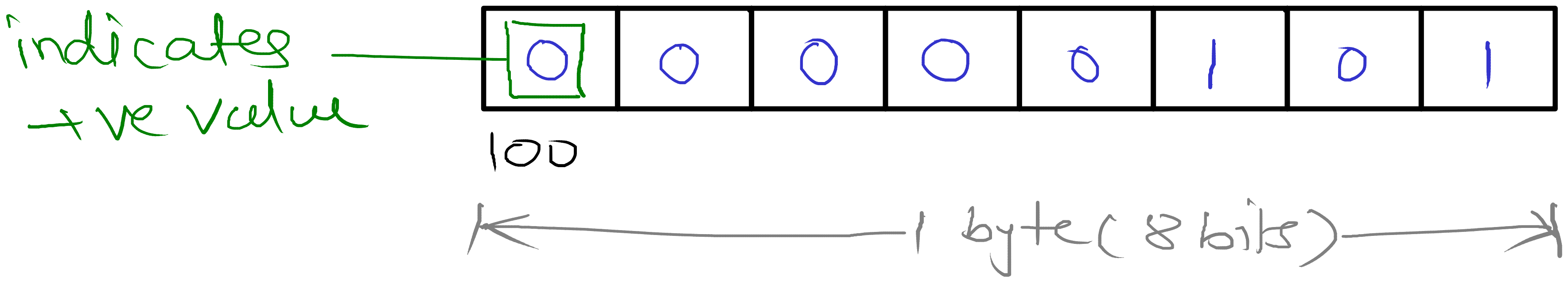
Signed vs Unsigned

signed char var = 5;



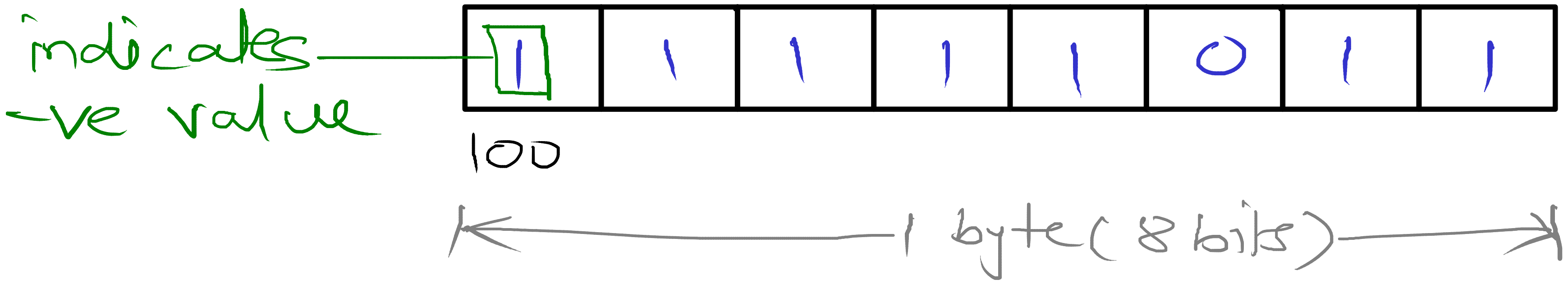
signed char var = -5;

5 = 0000 0101 (5)

1's complement = 1111 1010

+1 = + 1

2's complement 1111 1011 (-5)



Signed vs Unsigned

```
int main(void)
{
        signed char v1 = 5;
        signed char v2 = -5;
        if(v1 > v2)
                printf("v1 is greater\n");
        else
                printf("v2 is greater\n");
        return 0;
}

// result :- v1 is greater
```

v1 = 0000 0101 (5)

v2 = 1111 1011 (-5)

```
int main(void)
{
        unsigned char v1 = 5;
        unsigned char v2 = -5;
        if(v1 > v2)
                printf("v1 is greater\n");
        else
                printf("v2 is greater\n");
        return 0;
}

// result :- v2 is greater
```

v1 = 0000 0101 (5)

v2 = 1111 1011 (251)

Bitwise Operators

```
unsigned char v1 = 5, v2 = 3;
```

Bitwise AND (&)

```
    v1   =   5    0000 0101
&   v2   =   3    0000 0011
---------------------------------------
                  0000 0001 (1) (0x01)
```

Bitwise OR (|)

```
    v1   =   5    0000 0101
|   v2   =   3    0000 0011
---------------------------------------
                  0000 0111 (7) (0x07)
```

Bitwise NOT (~)

```
    v1   =   5    0000 0101
~
---------------------------------------
                  1111 1010 (250) (0xFA)
```

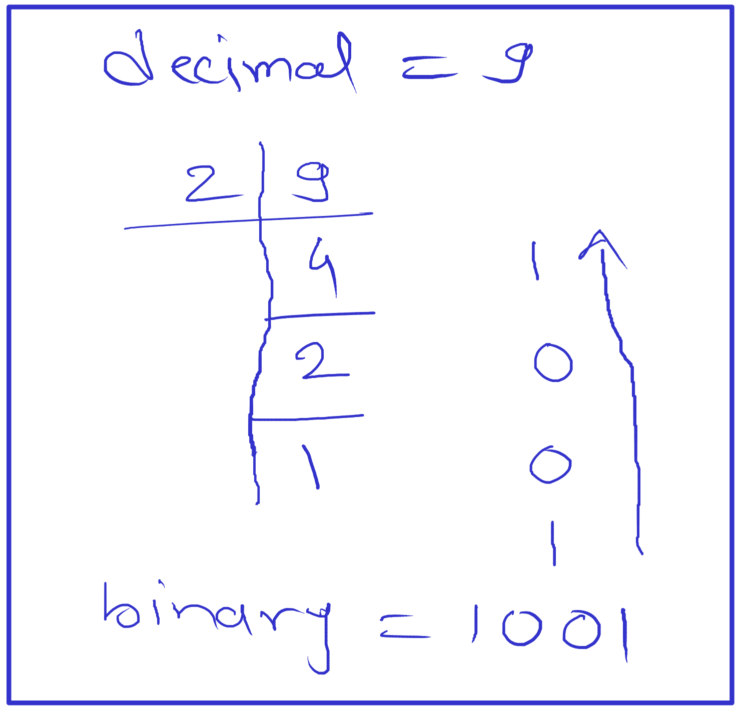
Bitwise XOR (^

```
    v1   =   5    0000 0101
^   v2   =   3    0000 0011
---------------------------------------
                  0000 0110 (6) (0x06)
```

HexaDecimal Number system

- base 16
- Number of symbols - 16   (0 to 9 and A(10) to F(15))

0	-	0000	
1	-	0001	
...			
7	-	0111	
8	-	1000	
9	-	1001	
10	-	1010	(A)
11	-	1011	(B)
...			
15	-	1111	(F)



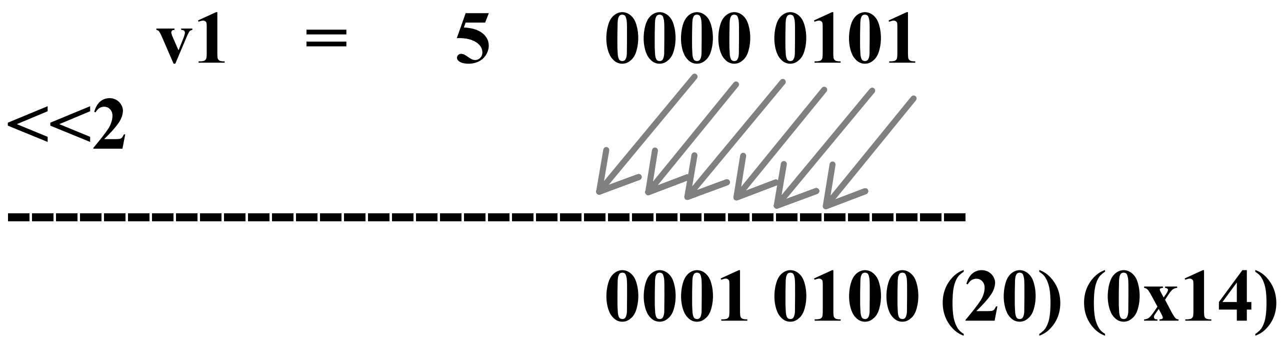
Any programming language

- we process data (values/numbers)
- numbers can be represented in decimal, octal, hexadecimal
- decimal   value
- octal   0value   - 0 indicates value is in octal
- hexa   0xvalue   -0x indicates value is in hexadecimal

Bitwise Operators

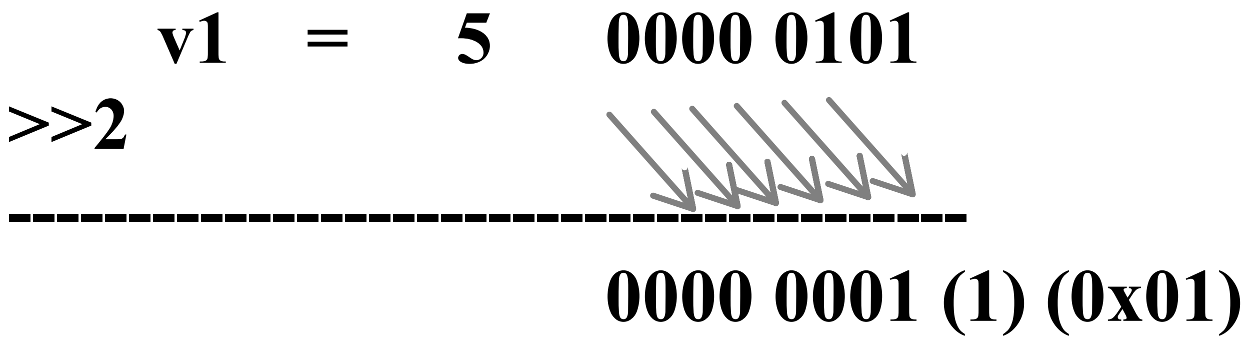
unsigned char v1 = 5;

Left shift (<<)



```
      v1    =     5      0000 0101
<<2
----------------------------------------
                         0001 0100 (20) (0x14)
```

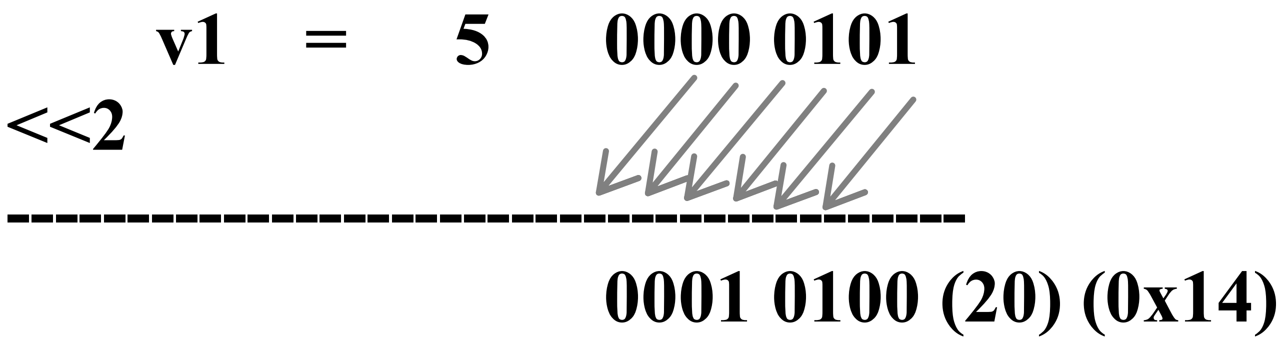
Right shift (>>)



```
      v1    =     5      0000 0101
>>2
----------------------------------------
                         0000 0001 (1) (0x01)
```

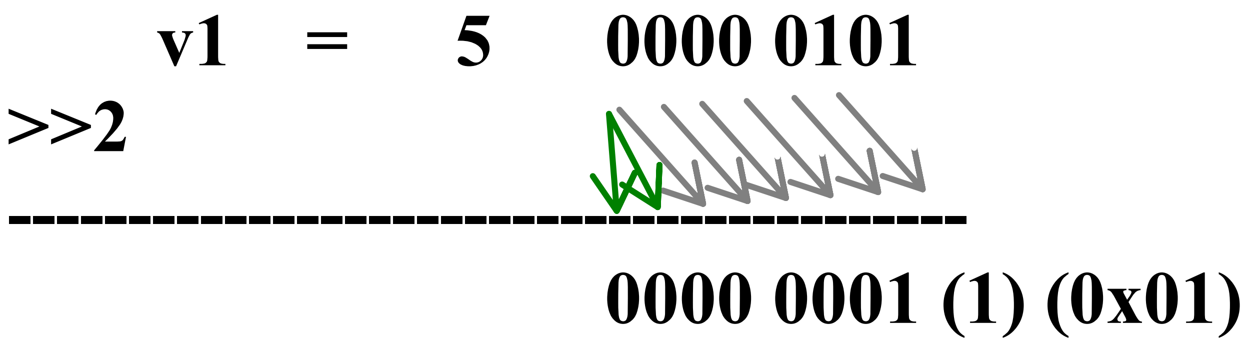
signed char v1 = 5;

Left shift (<<)



```
      v1    =     5      0000 0101
<<2
----------------------------------------
                         0001 0100 (20) (0x14)
```

Right shift (>>)



```
      v1    =     5      0000 0101
>>2
----------------------------------------
                         0000 0001 (1) (0x01)
```

Swap using bitwise operators

```
num1  =  5
num2  =  3

num1 = num1 ^ num2
                          0000 0101
                          0000 0011
                 num1 =   0000 0110
num2 = num1 ^ num2
                          0000 0110
                          0000 0011
                 num2 =   0000 0101
num1 = num1 ^ num2
                          0000 0110
                          0000 0101
                 num1 =   0000 0011


num1 = 3
num2 = 5
```

check even or odd

```
num = 1  0001       -- odd
num = 2  0010       -- even
num = 3  0011       -- odd
num = 4  0100       -- even
num = 5  0101       -- odd
num = 6  0110       -- even
num = 7 0111        -- odd


LSB = 0  --> even
LSB = 1  --> odd

     3     0011
&    1     0001
-------------------
           0001

     4     0100
&    1     0001
--------------------
           0000
```

Find number is divisible by 4 or not.

<< 1 ==> multiply 2
>> 1 ==> divide by 2

4	0100
8	1000
12	1100
16	0001 0000
20	0001 0100

5	0101
7	0111
9	1001
13	1101
18	0001 0010

Last two bits are 00 --> number is divisible by 4

```
    12   1100
&   3    0011
-------------------
         0000
```

12 is divisible by 4

```
    9    1001
&   3    0011
-------------------
         0001
```

9 is not divisible by 4

num = 10 ⇒ 0000 1010

mask = 0x80 ⇒ 1000 0000

num & mask ⇒ 0000 0000 → 0

mask >> 1 ⇒ 0100 0000

num & mask ⇒ 0000 0000 → 0

→ 0

→ 0

mask >> 1 ⇒ 0000 1000

num & mask ⇒ 0000 1000 → 1

→ 0

→ 1

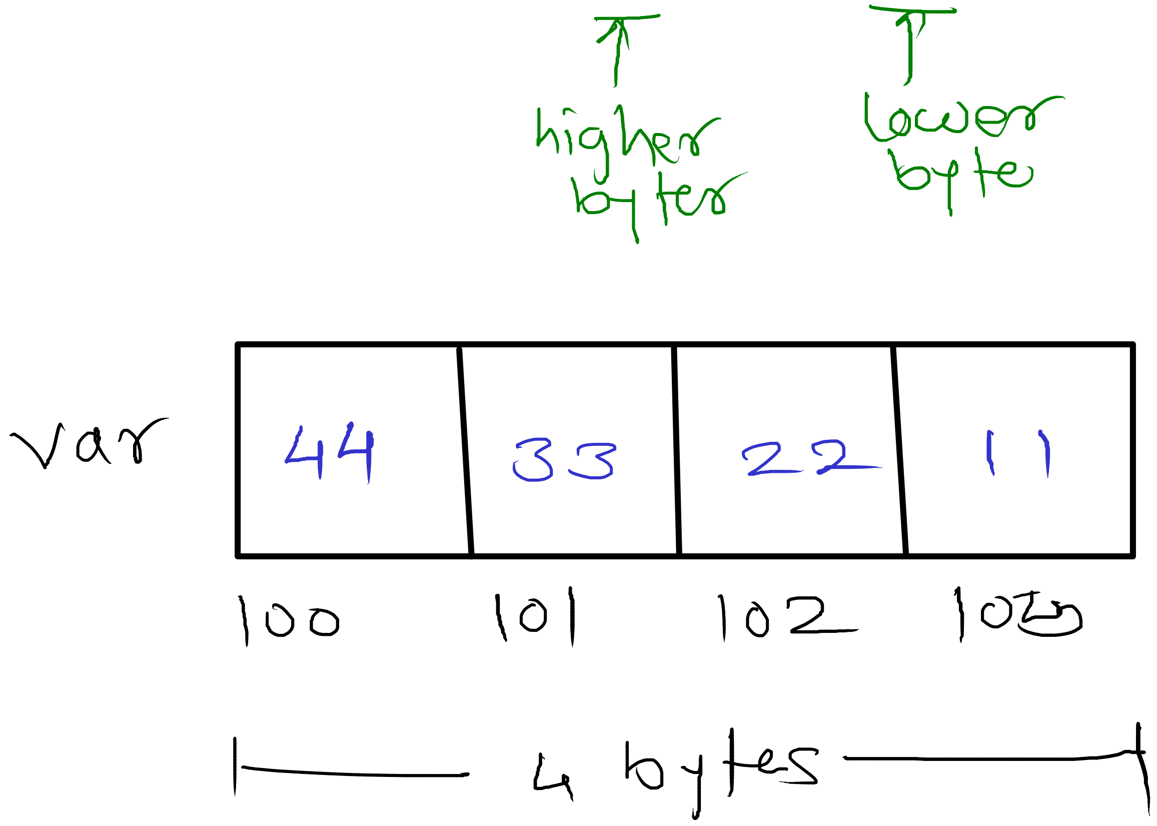
→ 0

mask >> 1 ⇒ 0000 0000 ✗

Endianness

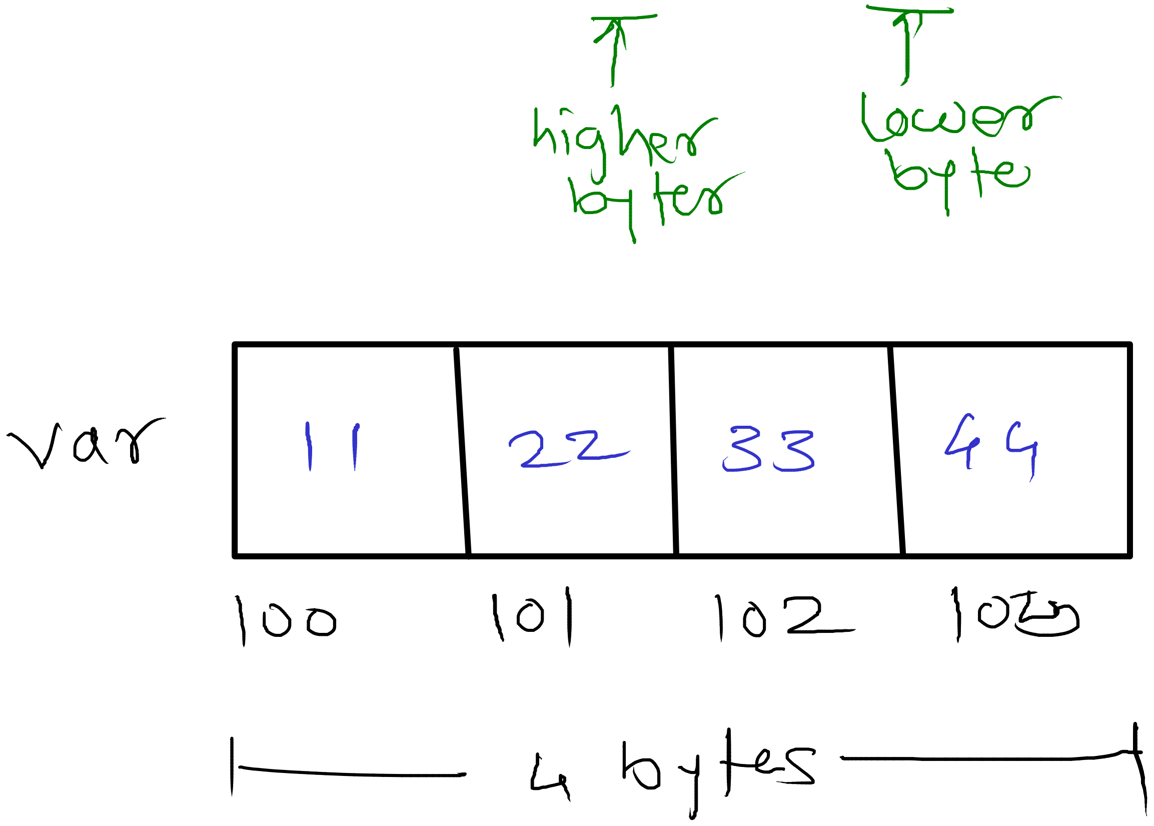
Little Endian

int var = 0x11 22 33 44;



Big Endian

int var = 0x11 22 33 44;



e.g. X86, AVR

ARM

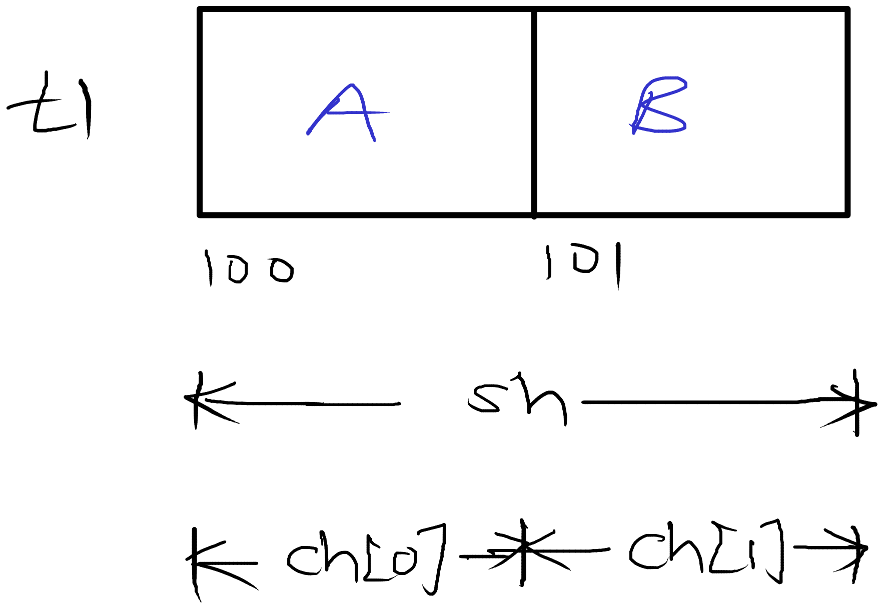
eg. PowerPC, Network

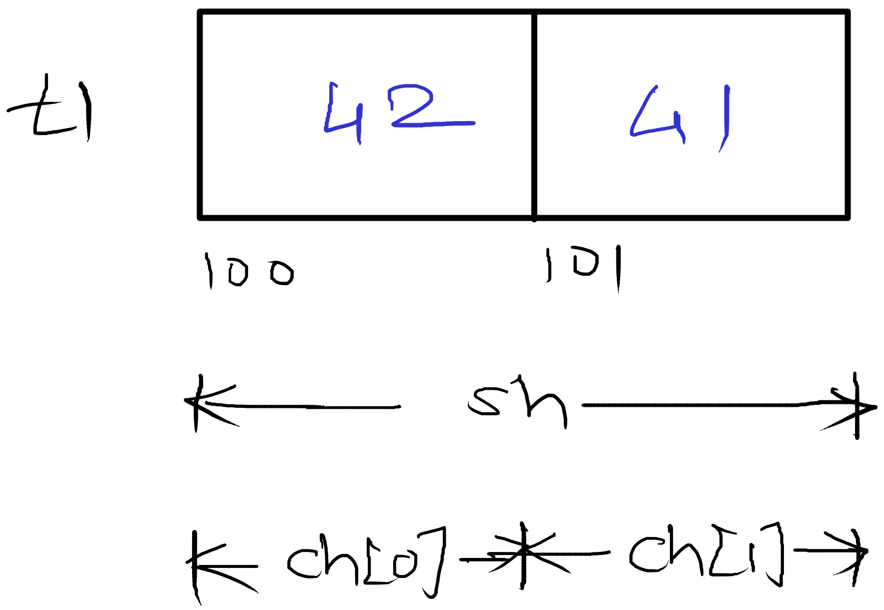
```
union test {
        char ch[2];
        short sh;

};
```

```
union test t1;

sizeof(t1) = 2 bytes
```

```
t1.ch[0] = 'A';
t1.ch[1] = 'B';
```

```
t1.sh = 0x4142
```





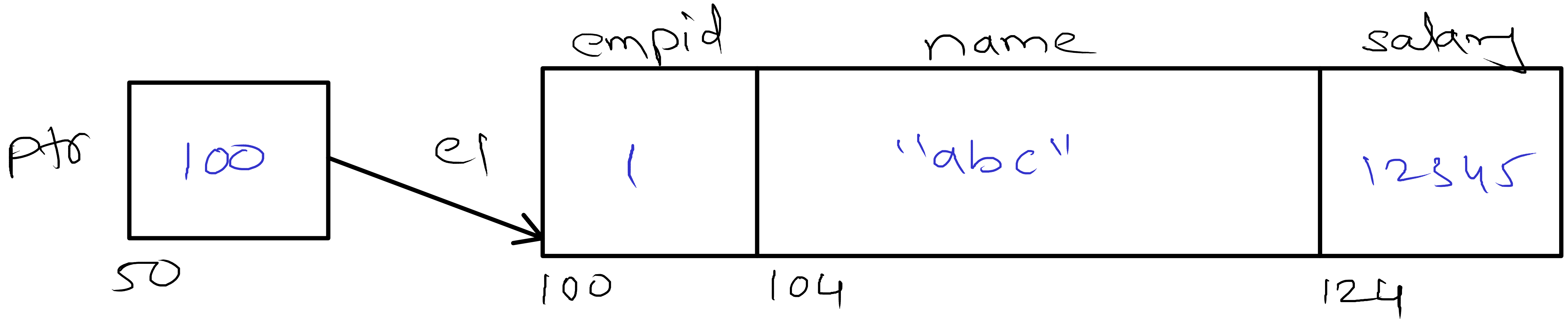
t1.sh = 0x4241

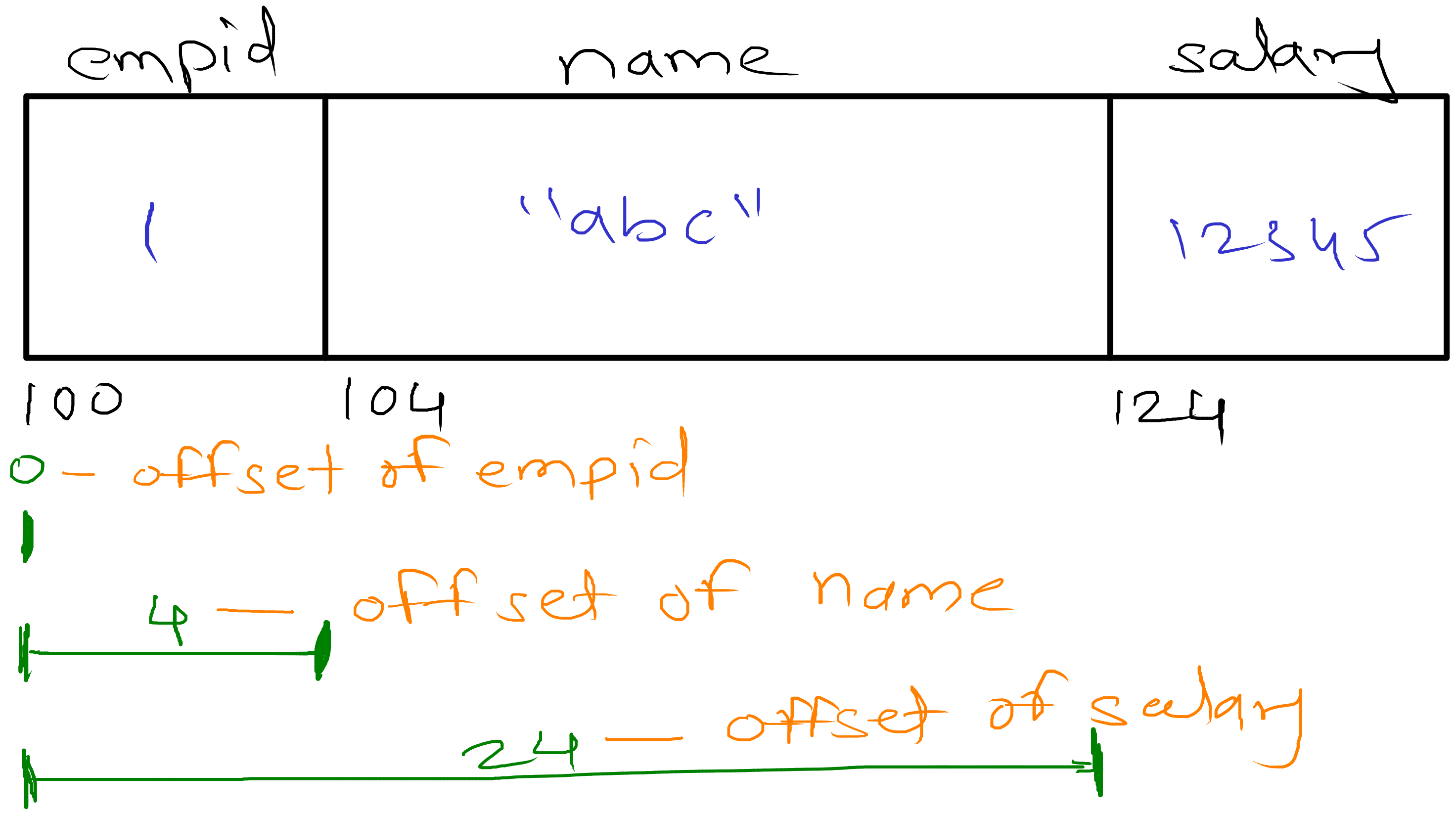
t1.ch[0] = 42
t1.ch[1] = 41

Structure Offset

```
struct emp {
      int empid;
      char name[20];
      float salary;
}e1;
```

```
struct emp *ptr = &e1;
```





e1 . <member>

ptr –> <member>

offset

↳ displacement of member from starting address of structure variable.