Type casting and Storing variables

conversion of data type to another (if it is valid) one is called Type casting.

Example1

```
int a = 'a';
```

variable a will store the ASCII value of 'a' = 97.

Example1

```
char ch = 98;
```

98 will automatically get typecasted to its corresponding character.

(a) This automatic typecasting is called implicit typecasting

```
int a = 'a';
cout << a << endl;
char ch = 98;
cout << ch << endl;
```

output:

97
b

⇒ what if we type cast from int to char but the value is too large to be stored in char?

Solution: A warning is thrown and last byte from the orginal value is given to the character

```
char ch = 123456;
cout << ch << endl
```

output

1 warning generated
@

⇒ How negative numbers are stored?

Solv: The first bit tells us whether the number is positive or negative

first bit → 0 means positive
↳ 1 means negative.

Steps to store -5 in binary format:

① Ignore the -ve sign. (5)

② write the binary representation of 5.

③ Take its 2's compliment and store it.

Example: a = -5

① -5 → 5 (Ignore the -ve sign)

② 5 → 0101 → 000... 0101 (Binary)
(29 zeros)

③ 2's compliment = 1's compliment + 1

5 → 000...0101

1's compliment → 11...1010 (Flip the bits)

\+ 1

2's compliment → 11...1011
(29 one)

⇒ Displaying Negative numbers!

① Take 2's compliment of the stored number

Stored: 1 11 ... 1011

↳ this shows -ve

↳ -5 print hoye gese!

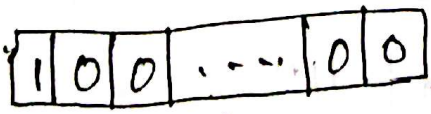
11..1011

1's → 00...0100

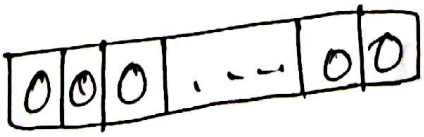
2's → 00...1011 = 5

② Why 2's compliment?

If we stored numbers as it is without using 2's compliment, then



and



will be equal and thus waste space

③ Store only positive Integers

The default signed representation allows us to store both positive and negative value.

To store only positive integers, we use unsigned.

Eg: unsigned int a = 10;

④ What if we store a negative number in an unsigned number?

Example!

```
unsigned int a = -112;
cout << a << endl;
```

output

4294967184 ??

Explanation!

we tried to store -112.

-112 = 2's compliment of 112

112 → 000...0110000
(24 zeros)

1's compliment → 111...1000 1111

+ 1

2's compliment → 11...10010000

unsigned int used all 32 bits to store the value and the MSB.(=1) will make the value. An unsigned int does not use the 2's compliment again to display the number.

Thus, 11...10010000. gets printed as it is in decimal

Therefore,

```
unsigned int a=112;
cout << a << endl;
```

→ output!
4294967184

Operators

Basic Arithmetic operators!

+, -, X, /, %

Caution ⚠