Programming Basics-I

⇒ int main() → main function from where code will execute or start.

⇒ #include <iostream> ← implementation is included in this.

inbuilt/standard

or

user created file.

⇒ using namespace std;

→ In every namespace a function eg cout has different implementation.

→ so we need "using namespace std" to use the current implementation in our code.

⇒ << → when we want to display on standard output.

⇒ endl → new line.

H/W → ① Can I create custom header file?

② Can I create our own namespace?

③ What all other namespace present instead of std?

Q) Can we print without/other than "<<"?

Data Types & Variable ?

4 bytes → content

int a = 5

1 byte → 8 bits

→ type of data
size of data

variable → name of block
created in memory.

bool a = true;
↓
1 byte → smallest addressable size.

float f = 1.2;
↓
4 byte

double d = 1.23
↓
8 byte

Double → has better precision. ~~and~~

Variable naming convention -

- → small, capital letters. {abc}
- → include numbers. {babbar1}
- → {a_b} underscore allowed.
- → cant include a no. in start {1abc}

H/W explore about short and long?

How ~~is~~ are contents stored in memory?

int a = 5 → binary → 101

variable → a
~~byte~~ → int
size → 4 byte → 32 bits

00000000	00000000	00000000	00000101
0	0	0	5

This is applicable for + numbers only.

⇒ How -ve numbers are stored in memory?

int x = -5

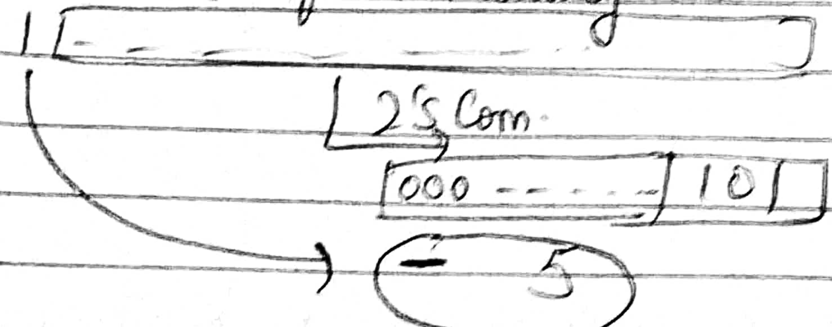
Algorithm -
↳ ignore -ve sign.
↳ Convert into Binary rep.
↳ take 2's Compliment

5
↓
$\underbrace{0000 \ldots 101}_{32\ bit}$

2's Compliment
↳ take 1's Compliment +1
↓
flip all bits
0 → 1
1 → 0

```
00000000    00000000    00000000    00000101
 ↓ 1's C     ↓ 1's C     ↓ 1's C     ↓ 1's C
11111111    11111111    11111111    11111010
                                          + 1
11111111    11111111    11111111    11111011
```

To access from memory –



H/W.

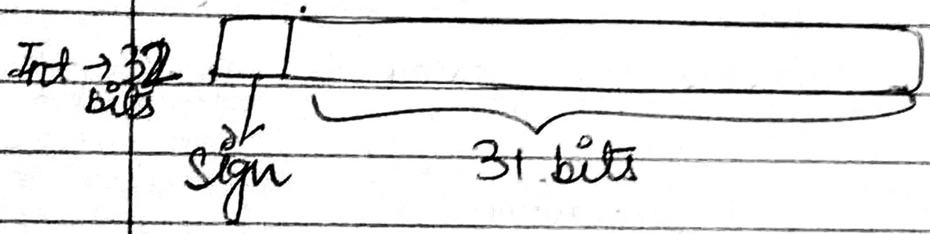
Dry Run –

-11

-4

-8

-5



If I have 31 bits, how much no's I can create with this?

Range → $-2^{31} \longrightarrow 2^{31}-1$

→ char → 1 byte → 8 bits

$2^8 \to 256$

char is stored in the form of ASCII values.

Every character is ~~stored~~ ~~as~~ associated to an integer number and int is stored in memory in the form of binary.

Q. How are we going to differentiate b/w int or char in memory, since both get stored as 0 or 1?

Ans. Datatype tells whether its a char or int.

H/w Range of -
↳ float →
→ double →
→ long →
→ short →

Operators :

① Arithmetic Operators – +, -, *, /, %
mathematical operations.

int ans = a/b ← int/int
↑ it will store int value.

$\frac{\text{float}}{\text{int}} \rightarrow \frac{5.0}{3} \rightarrow 1.6$ storing in int → int ans → 1.

```
int ans = 5.0/3;
cout << ans;   ← X
```

$\boxed{X = 1}$

```
cout << (5.0/3);   ← Y
```

Y = 1. ... {in decimal}
↳ float to int → float

int → int
int

float → float
int

double → double
int

Typecasting → ① **Implicit** - Compiler automatically converts into required datatype

② **Explicit** - forcefully converted.

```
char ch = 'a';
int num = (int) ch ;
```

} output: 97
↳ ascii value of 97

```
char ch = 'b';
int num = (int) ch ;
```

} output: 98

② Relational Operator :

== > < >= <= !=

== → comparison

a == b

false   true

bool b = (x == y)

0

false

x = 5, y = 3

= → assignment operator

③ Logical Operators:

&&	AND
||	OR
!	NOT

(&&) → both conditions should ~~[illegible]~~ satisfy (true) to get true.

bool ans = ( ) && ( )

T → T T
F → F T
F → T F
F → F F

(||) → any one condition needs to be true.

bool ans = ( ) || ( ) || ( )
T T F F

(!) → compliments the value.
1→0
0→1.

④ Bitwise Operators:

↳ Bit level → &

(i) & int a = 5 → 101
int b = 6 → 110
int ans = a & b
↳ 4

$$\begin{array}{r} 101 \\ 110 \\ \hline 100 \end{array} \rightarrow 4$$

ii) OR → | a=5   101

ans = a|b   b=6   110

ans = 7   (7) 111

iii) NOT ~ tilda

↳ 0 → 1

1 → 0

(iv) XOR → Exclusive OR → very important

(^)

x	y	O/P
0	0	0
0	1	1
1	0	1
1	1	0

H/W Arithmetic, Logical, Relational, Bitwise } Experiment, Code, Explore } On code editor

→ Left Shift operator :-

$\boxed{5<<1}$ → shift 5, by 1 bit

000000000101   5<<1 → 10

00000001010
↳10

5<<2
shift 5 by 2 bits

000000101 ← 2

0000010100 ← 20

whenever we shift a no. by left shift, we are multiplying it by 2 (but not always).

1 → 10 , 2 → 20

[0| 1 1 1 1 1 1 1 1 1 1 1 1] → +ve

1 bit

[1| 1 1 1 1 1 1 1 1 1 1 1 1 1]

↳ -ve fas gye

→ Right shift operator :- number % 2 → but not always.

5 >> 1

00000101 $\quad 5/2 \to 2$

00000010 → 2

5 >> 2

00000101 → 1 $\quad \frac{5}{2\times2} = \frac{5}{4} \to 1$

00000001 → 1

H/W How << and >> work on -ve numbers?

In left shift, we add a zero on right side → this is called padding.

- → +ve no. → padding is done by adding 0.
- → -ve → padding is compiler dependent.