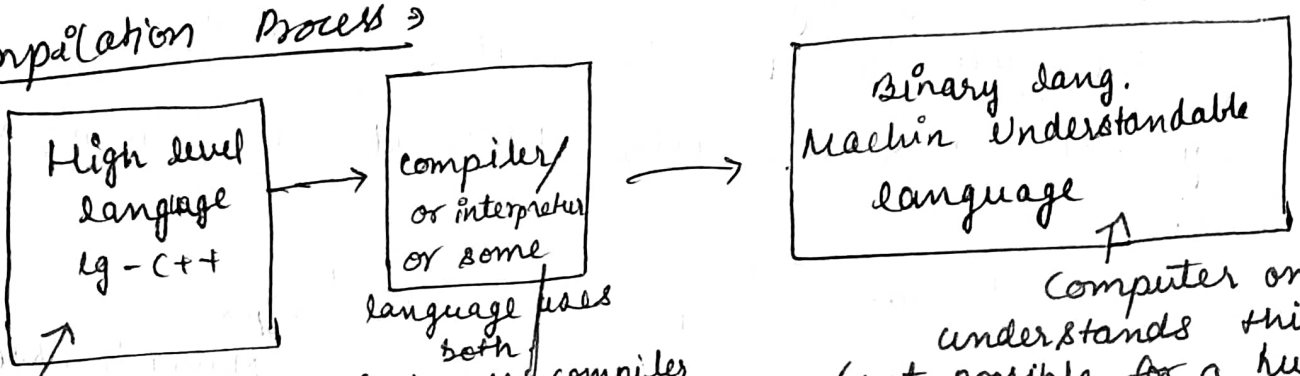
28 Jan

Lec-2 **Create your first program & a lot more**

① Compilation Process and Programming Language →

⊙ Compilation Process →



we write this, not understandable by the system.

language uses both C++ uses compiler

Compiler converts the high level lang. to binary lang.

Computer only understands this (not possible for a human to write each program in m/c lang.)

⊙ Programming Languages → language in which we are giving instructions to our computer.

→ Every lang. has it's own compiler / Interpreter.

Eg → C, C++, Java, Python.

H.W → Diff. b/w Complier and Interpreter. Explore them.

⊙ IDE (Integrated Development Environment) →

helps us to write code and execute them and provides many additional features.

Eg → VS code, Code Blocks, Sublime etc.

⊙ Let's code our first program → "Namaste Bharat"

code between these curly braces is under main() function.

```
int main(){ ← Execution starts from here always.
    cout << "Namaste Bharat";
}
```

cout is used to print (in C++)

We are getting an error (use of undeclared identifier)

Because the code of cout also has written somewhere so we must import that. To import the file →

```
#include <iostream>
```

Still we are getting same error and a suggestion did you mean "std::cout".

By replacing cout<< with std::cout our code works fine.

```
#include <iostream>
int main(){
    std::cout<<"Namaste Bharat";
}
```

o/p→ Namaste Bharat

std is a namespace. Namespace is a particular region where scope of identifiers is defined.
We can write like this

using namespace std; (before main()).

Then we don't have to write it everytime.

In order to print something we must use cout. and with cout we use "<<" (insertion operator).

```
cout<<"Hi";
```

cout → to print
<< → insertion operator (to print onto standard display)
"Hi" → string
; → semicolon shows end of line.

Now the code→

```
#include <iostream>
using namespace std;
int main(){
    cout<<"Namaste Bharat";
}
```

```
cout<<"2"; → 2 as a string
cout<<2; integer 2
cout<<'a'; character
```

- endl → endl is used to print a line.
- \n → same works as endl.

⊙ How to take input from the user →

Let's see how to comment?

1→ // This is a comment

2→ /* This is
a multiline
comment */

→ To take a number in input we have to store it (otherwise how will we use it (if needed)).

```
int a;          // That storage is named as a.
cin >> a;       // Taking input.
cout << "You entered " << a << endl;
```

let say i have input as 5.

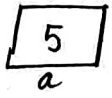
o/p → You entered 7

⊙ Datatypes & Variables →

Variable - named memory location.

```
int a = 5;
```

a is a variable.



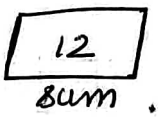
Datatype - It is used to tell the variable the type of data they can store. `int a = 5;` int is a datatype here.

eg;→ char, short, int etc.

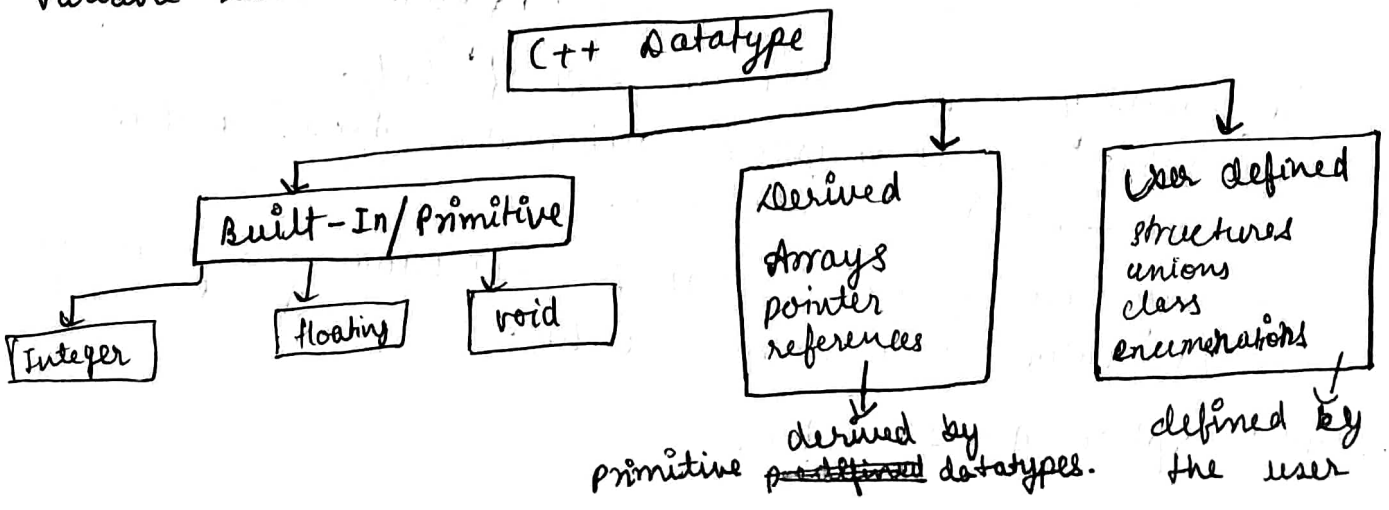
```
int sum = 12;
```

value

datatype — variablename



This lines means integer type of value will be stored in variable sum and it's value is 12.



~~Function~~

```
int main(){
  ---
  ---
}
```

return type

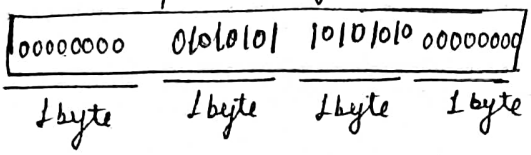
means func will return an integer.

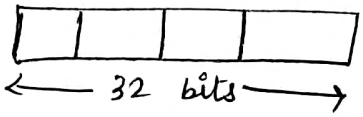
```
void main(){
  ---
  ---
}
```

return type

This func will return nothing.

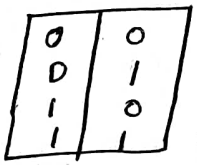
→ integer is of 4 bytes (32 bits). (or maybe 2 bytes)
It depends upon system architecture.





⟹ means 2^{32} combinations.

let's suppose we have 2 bits.
so possible combinations



4 combinations $= 2^2$

→ Char is of 1 bytes.

1 bytes = 8 bits.

Total combinations $= 2^8 = 256$

```
char ch = 'a';
```

ch [a]

we can fill 0 or 1 here

so total 2 combination for 1 bit.

for 8 bits combinations $= 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8$
$= 256$

In memory it is stored in form of bits.
Each character is mapped with an ASCII value
(0 – 255) ⟶ Total 256.

So they are stored in form of ASCII values. For eg a = 97.
ASCII value is a numeric value.

↑ ASCII value of a

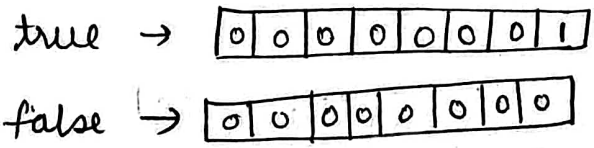
H·W → Explore ASCII Table.

→ bool flag = true;

true means 1.
false means 0.

bool store either true (1) or false (0).
bool takes space of 1 bytes. Although 1 bit is sufficient to represent.

⊙ Why is take 1 byte and not 1 bit?
⇒ Because 1 bit is not addressable unit, 1 Byte is minimum addressble unit



So yes there is wastage of memory.

→ Float →

eg- float f = 1.2;

float and double both are used to store floating points.

float = 4 byte → 32 bits
double = 8 byte → 64 bits

double is more precised than float.
Storage is different of float and double.

⊙ What happen when we try to print 256 (by character datatype)

```
int ch = 256;
cout << ch;
```

char's range is of (0-255) or (-128 to 127) now if we tried to access 256, it is overflow condition. And compiler may behave differently in this case. (Maybe it can print a garbage value or nothing).

H.W → variable naming conventions.

⑤ How to see size a variable is taking?

```
int a = 5;
cout << sizeof(a);          // 4 or 2
char c = 'q';
cout << sizeof(c);          // 1
double d = 5;
cout << sizeof(d);          // 8
bool flag = 1;
cout << sizeof(flag);       // 1
```

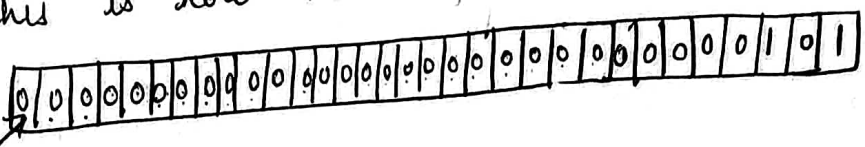
⑥ How data is stored?

- positive integers →

int a = 5;

it's binary ~~representation~~ representation = 101

so this is how it is stored.



first bit is 0

so the first bit of a positive number is 0.

and the first bit of a negative number is 1.

- negative no. storage → Negative numbers are stored in the form of 2's complement.

2's complement ⇒ 1's complement + 1

1's complement ⇒ flip the bits.

for eg → 7

7 → 00000111 (for understanding we took 8 bits, actually there should be 32 bits.)

1's complement → 11111000

2's complement → 11111000 + 1 = 11111001

let's store -5. a [5]

steps ⇒

① Ignore -ve sign [5]

② Find binary equivalent

000 ___ ___ - 101

← 29 bits →

③ Find 2's complement.

1's complement → 111----1010
(29 bits)

2's complement → 111------1011
(29 bits)

so this is how it is stored in memory.

Now how to read it from memory.

⇒ Take 2's complement.

2's complement of (1)----1011 → (29 bits)
(1) → if shows -ve.

1's complement + 1 = 0---0100. +1

2's complement = 0-----0101 = 5

and the sign will be negative.

so -5.

Interesting Problem →

How will we know that we have to read only 1 bytes or 4 bytes or 8 bytes from starting of a memory block?

⇒ By datatype.

datatype defines two things.

↳ which type of data will be stored?

↳ How much space it will take?

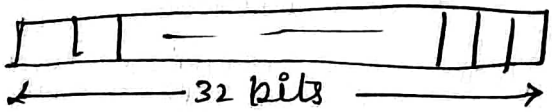
⊙ Signed v/s Unsigned data →

signed → -ve, 0, +ve

unsigned → +ve, 0

By default data is signed.

eg, int → 4 bytes = 32 bits



Total combination = 2^{32}

Total addressable range →

for signed → [redacted] -2^{31} to $2^{31}-1$

for unsigned → [redacted] 0 to $2^{32}-1$

short → 2 bytes → 16 bits

Total combination → 2^{16}

unsigned → 0 → $2^{16}-1$

signed → -2^{15} to $2^{15}-1$

char → 1 byte → 8 bits

Total comb → 2^8

unsigned → 0 to 2^8-1

signed → -2^7 to 2^7-1

xyz → 6 Bytes → 48 bits

Total comb. → 2^{48}

unsigned → 0 to $2^{48}-1$

signed → -2^{47} to $2^{47}-1$

So General formula →

if we have n bits, total comb = 2^n.

So unsigned → 0 to 2^n-1

signed → -2^{n-1} to $2^{n-1}-1$

⊙ **Typecasting** → It refers to the conversion of one data type to another. Two ways →

(Implicit) → Automatically

(Explicit) → Manually.

→
```
char ch = 97;
cout << ch << endl;
```
o/p ⇒ a.

we gave integer as input but we got a (char).

So integer is typecasted in character.

```
int num = b;
cout << num << endl;
```
o/p ⇒ 98.

When the typecasting is done automatically by the compiler it is called Implicit Typecasting (or type conversion).

→
```
double d = 5.7;
int x = (int) d + 2;
cout << x << endl;
```
o/p ⇒ 7 (5+2 = 7)

5.7 is typecasted to 5 manually, it is called Explicit.

Type casting or implicit type conversion.

⊙ Operators →

↳ Arithmetic (+, -, *, /, %)
↳ Relational (>, <, >=, <=, !=, ==)
↳ Assignment (=
↳ Logical
↳ Bitwise

A doubt → What happens when we give char as 1024

char ch = 1024;

ch → 1 byte → 8 bits
1024 → 16 bits

0 ------- 10000000000

Maybe it will take the last 8 bits or.
overflow or
Garbage value.
or error.

• Arithmetic Operator →

int a = 5, b = 3;
cout << a+b << a-b << a*b << a/b << a%b;

a+b = 8
a-b = 2
a*b = 15
a/b = 1
a%b = 2

division →

int / int = int,
float / int = float,
double / int = double,
int / double = double.

H.W ⇒ Explore precedence table

• Relational operator → output as true or false.

int a = 5, b = 3
cout << (a > b); // o/p → 1
cout << (a < b); // 0
cout << (a == b); // 0

```
cout << (a!=b);          // 1
cout << (a>=b);          // 1
cout << (a<=b);          // 0
```

- **Assignment Operator** – Used to assign value to a variable
  `int a = 5;`

- **Logical Operator** – When we have multiple conditions to decide an output.

To vote → your age must be 18 or greater than that and you must be a citizen of India.

AND →

⇒ `cout << (age >= 18 && citizen == India)`

o/p will be 1 only both condition are true.

OR →

```
int a = 5, b = 3
cout << (a<=5 (T) || b<=3 (T))      // 1
cout << (a<5 (F) && b>=3 (T))       // 0
cout << (a<5 (F) || b>=3 (T))       // 1
cout << (a>5 (F) && b<3 (F))        // F
```

|| → OR
&& → AND
! → NOT

```
cout << !(a>=5);   // a>=5 is T ⇒ !T = F
```

→ `(cond1 && cond2 && cond3)`

let say this is False.

Then our compiler is not going to check other two conditions. (Because in AND all values must be true in order to get the o/p as true).