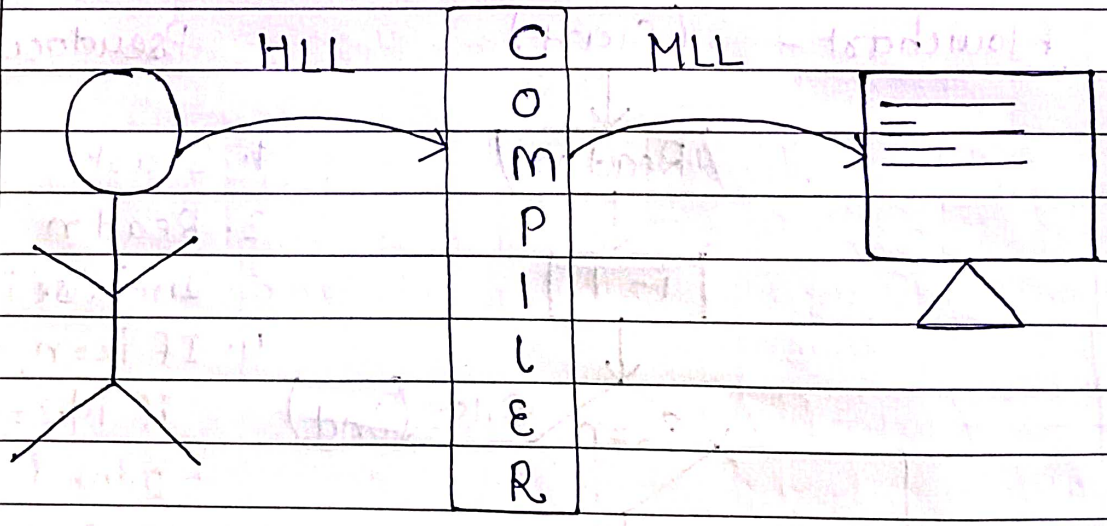
25/08/2023
Friday

* PROGRAMMING LANGUAGES -

- A language using which we can instruct the computer to carry out real life tasks and computations is called programming language. It acts as a language in which we could easily express our thoughts to the machine.
- It has a fixed set of rules. These programs are then converted into a language which machines can understand. These task is carried out by compiler.
- Every language has its own compiler / interpreter.

* Compilation Process -



Compiler acts as a translator.

__/__/____

* **WHERE TO CODE –** Code Editors or IDE's.

IDE stands for Integrated Development Environment.

CODE EDITORS –
1. VS Code
2. Sublime
3. Code :: Blocks
4. XCode
5. Eclipse
6. Atom
7. Online Compilers and many more.

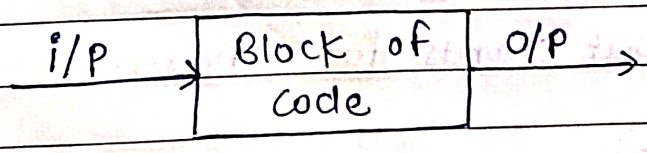
* **FIRST CODE –**

• **Starting of a code –**
```
int main( ){

}
```

1. int = return_type.
2. main( ) = function_name.
3. { } = Curly Braces defines the scope of main function

function – function is a block of code to which we give some input and it may or may not return the output. It performs a specific task.



NOTE - A running program is known as process. During running a program, we create a process and in the memory of that process iostream file must be included.

__/__/____

- **HEADER FILE -**

```
# include <iostream>
```

1. #include = Preprocessor directive. It is a way of including a standard or user-defined file in the program.

2. iostream = Iostream stands for standard input-output stream. This file contains definitions of objects like cin, cout etc.

- **NAMESPACES -** We can create our own custom namespaces. Namespaces are just a portion of code.

```
using namespace std;
```

1. namespace = namespaces are used to prevent name collisions.

2. std = std stands for standard.

i.e we have to use the standard namespace

- **Cout -** cout stands for character output.

```
cout << "string_name";
```

1. Cout = Used for displaying or printing the output on the console window.

__/__/____

2. << = Insertion operator

3. "string-name" = The text inside inverted commas is a string which we have to display.

4. Semi-colon (;) = terminator

- **End line or new line character –**

endl = stands for end line.
'\n' = new line character.

Both are used to go to the next line.

- **return 0; –**

It means that the program is executed successfully. It is totally optional to type return 0 at the end. Absence of this statement will never raise compiling error. By default, return 0 is always added.

PROGRAM –

```
#include <iostream>
using namespace std;

int main ()
{
  cout<< "NISHA" << endl;
  return 0;
}
```

__/__/____

* VARIABLES –

Variable is a name given to a memory location. It is the basic unit of storage in a program. The value stored in a variable can be changed during program execution. All the operation done on the variable effects that memory location.

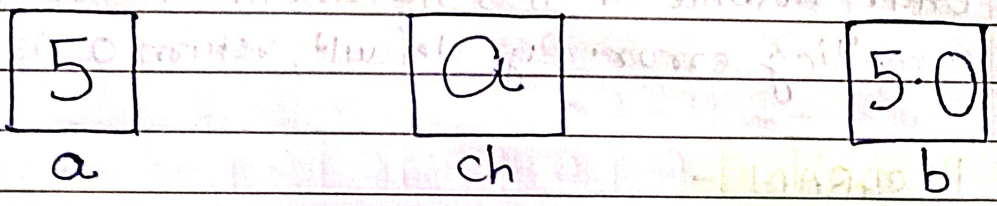
* DATATYPES –

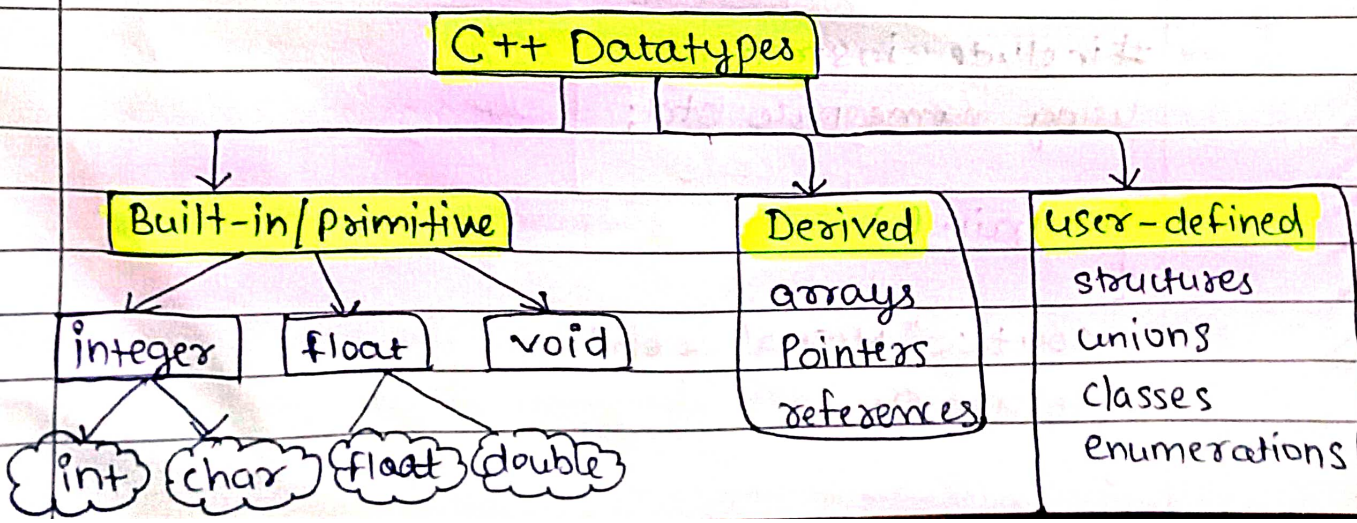
The type of value whether int, char, float, double etc. stored in the variable is defined by the Datatype.

- Data type tells two things → 1. the type of value stored
  2. Size or memory it will take

For instance →

int a = 5; char ch = 'a'; float b = 5.0;





NOTE - Operator sizeof( ) is used to find how much memory is allocated to which data type.

__/__/__

Declaration - int number;

Initialisation - int num = 50;

This shows how declaration and initialisation is done.

* **C++ Basic Data types**

		Size (Bytes) 32-bit CPU	Size (Bytes) 64-bit CPU
1.	char	1	1
2.	short	2	2
3.	int	4	4
4.	long	4	8
5.	longlong	8	8
6.	float	4	4
7.	double	8	8

* **VARIABLE NAMING CONVENTIONS -**

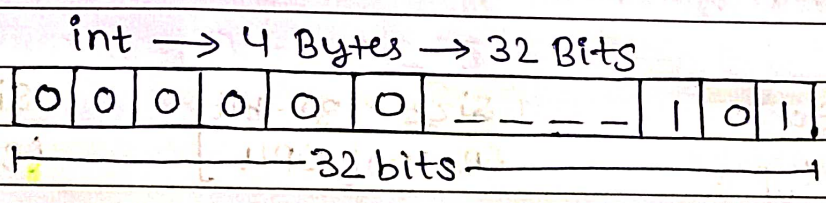
- It should begin with an alphabet.
- There may be more than one alphabet, but without any space between them.
- Digits may be used but only after alphabet.
- No special symbols can be used except underscore (_).
- No keywords or commands can be used as variable name.
- All statements in C++ language are case sensitive.

__/__/____

* **HOW DATA IS STORED? +ve and -ve numbers**

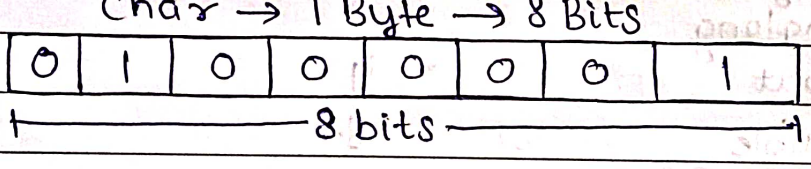
for instance,

→ **int a = 5;**



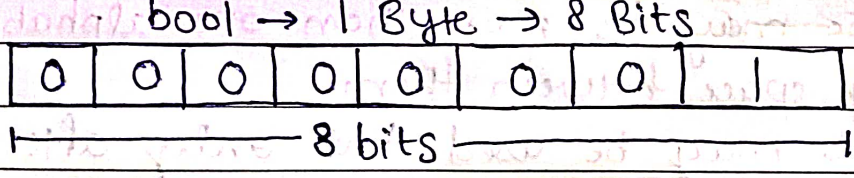
The binary equivalent of 5 is stored in the memory

→ **char ch = 'A';** [Ascii value of A = 65]



The binary equivalent of ASCII value of 'A' is stored in the memory.

→ **bool flag = true ;** [true = 1 , false = 0]



The binary equivalent of true i.e. 1 is stored in the memory.

NOTE - Bool datatype stores true→1 and false→0. It can be done or stored using 1-bit. But it is not possible because smallest addressable space in the memory is 1 Byte.

__/__/__

1's complement - Flip 1 and 0.

Eg - 1001110

1's complement - 0110001

2's complement - Step 1 → Find 1's complement.

Step 2 → Add 1 to 1's complement.

Eg - 1000

1's → 0111

2's → 0111

+ 1

1000

* Steps to store -ve number →

Step 1 - Ignore -ve sign.

Step 2 - Find Binary equivalent.

Step 3 - Take 2's complement.

For instance,

→ int a = -5;

Ignore -ve sign.

Binary equivalent = 00000000 00000000 00000000 00000101

2's complement = 11111111 11111111 11111111 11111010

+ 1

Memory → 11111111 11111111 11111111 11111011

__/__/__

* **RANGES OF DATA TYPES - FOR UNSIGNED DATA**

1. **char** → 1 byte → 8 bits

Total combinations = 2^8

> As these are 8 blocks and each block can hold two values i.e. 0 and 1.

Range = 0 ⟶ 255

or 0 ⟶ $2^8 - 1$

2. **int** → 4 bytes → 32 bits

Total combinations = 2^{32}

Range = 0 ⟶ $2^{32} - 1$

0 ⟶ $2^n - 1$ (in general)

3. **long** → 8 bytes → 64 bits

Total combinations = 2^{64}

Range = 0 ⟶ $2^{64} - 1$

NOTE - In general, if a data type has n-bits.

So, total combinations = 2^n

Range = 0 ⟶ $2^n - 1$

* **SIGNED VS UNSIGNED DATA -**

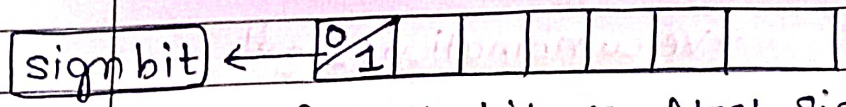
- **Signed Data** → It can hold positive, negative numbers, zero i.e. all integer values.

- **Unsigned Data** → It can only hold positive numbers and zero.

__/__/____

* RANGES OF DATA TYPES- FOR SIGNED DATA

\# To find whether the stored number is +ve or -ve.

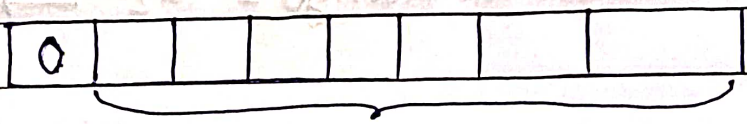


The leftmost bit or Most significant bit (MSB) tells whether the stored number is +ve or -ve.

If 0 → +ve integer

1 → -ve integer

1. char → 1 byte → 8 bits

+ve =  Bits left = 7

+ve combinations = 2^7

-ve =  Bits left = 7

-ve combinations = 2^7

+ve Range = 0 → $2^7 - 1$

0 → 127

-ve Range = -1 → -2^7

-1 → -128

0 is not considered as already considered in +ve Range.

Combined Range = -128 → 127

2. int → 4 bytes → 32 bits

+ve = 

+ve combinations = 2^{31}

-ve = 

-ve combinations = 2^{31}

+ve Range = $0 \longrightarrow 2^{31} - 1$

-ve Range = $-1 \longrightarrow -2^{31}$

Combined Range = $-2^{31} \longrightarrow 2^{31} - 1$

NOTE - In general, if a data type has n-bits.

Then,

+ve and -ve combinations = 2^{n-1}

+ve Range = $0 \longrightarrow 2^{n-1} - 1$

-ve Range = $-1 \longrightarrow -2^{n-1}$

Combined Range = $-2^{n-1} \longrightarrow 2^{n-1} - 1$

__/__/____

* **OPERATORS –**

1. Arithmetic – (a) addition [+] (b) subtraction [-]
   (c) multiplication [*] (d) division [/]
   (e) modulus [%]

2. Relational – (a) Greater than [>] (b) less than [<]
   (c) Greater than equal to [>=]
   (d) less than equal to [<=]
   (e) not equal to [!=]
   (f) equal to [==]

3. Assignment – [=]

4. Logical – (a) logical AND (&&) | used to check upon
   (b) logical OR (||) | multiple conditions
   (c) logical NOT (!)

5. Bitwise

HOW TO TAKE INPUT ?

* **cin** – cin stands for character input.

```
cin >> xyz ;
```

1. cin = cin is used to accept the input from the standard input device i.e. keyboard.

2. >> = Extraction operator.