04-06-2025 WEDNESDAY

EMBEDDED SYSTEM

↓

Hardware + Software
↳ Firmware

Microcontroller | Microprocessor | System On Chip (SOC's)
↓
(Instructions)

- Microcontroller
  - → Washing Machines
  - → Micro Oven
  - → A.C.
  - → T.V. Remote

- SOC's
  - → T.V.
  - → Smartphone
  - → Surveilance camera

- Microprocessor
  - → CT scanner
  - → Drones
  - → PC's, Laptops

Microcontroller

- Single core
- Speed

1) 8051 → 12MHz
↓
cycles/sec

μC speed is measured in Hz (Cycles/sec)

- 8051 is designed by intel in 1980
- Advanced versions of 8051 are 22MHz to 80MHz

2) Arduino is a hardware & software platform.
→ ATmega2560 μc

- ATmega2560 μc is developed by Atmel in 1998.

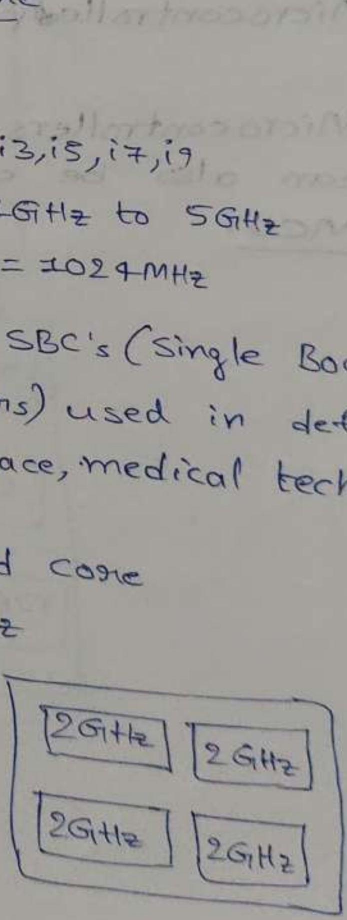
Microprocessor

- Multi core
- Speed

1) Intel i3, i5, i7, i9
Above 1GHz to 5GHz
1GHz = 1024MHz

2) Intel SBC's (Single Board Computers) used in defence, aerospace, medical technology

Quad core
2GHz



- ATmega2650 operates at a max. clock speed of 16MHz

3) ARM7 → 12MHz

- Advanced versions runs at 88MHz
- Developed by ARM Holdings in 1990

Most of the μc runs below 100MHz

05-06-2025

Thursday

- Memory

Memory required for μc is very less (i.e., KB's to MB's)

- OS

No OS

*Note:- We can't run a full fledged Embedded OS on a Microcontroller based device

- Microcontrollers in short can also be called as MCU.

- Memory

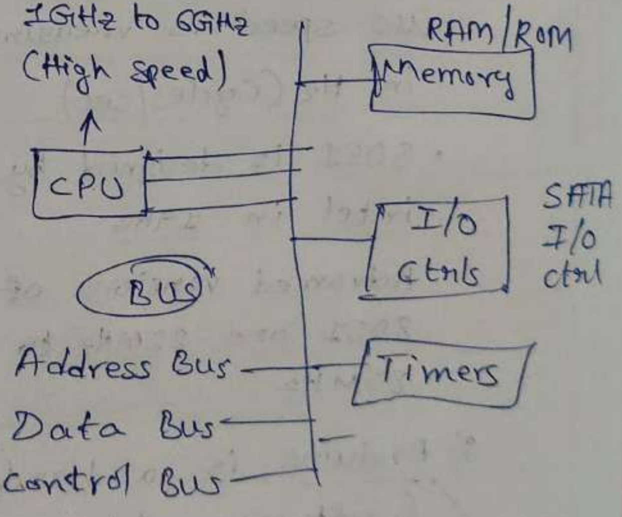
Memory required for μp is MB's to GB's

- OS

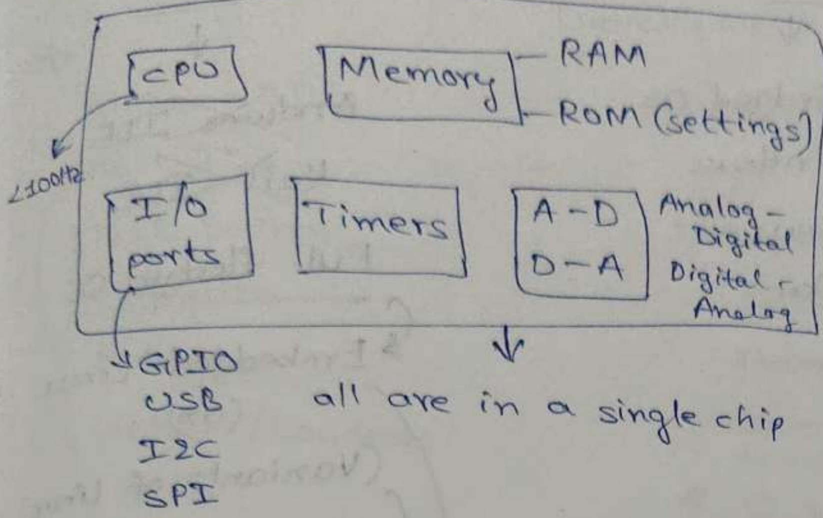
Windows, Linux, Mac

- Hardware components like memory, I/O controllers & timers e.t.c. are connected through physical lines called BUS.

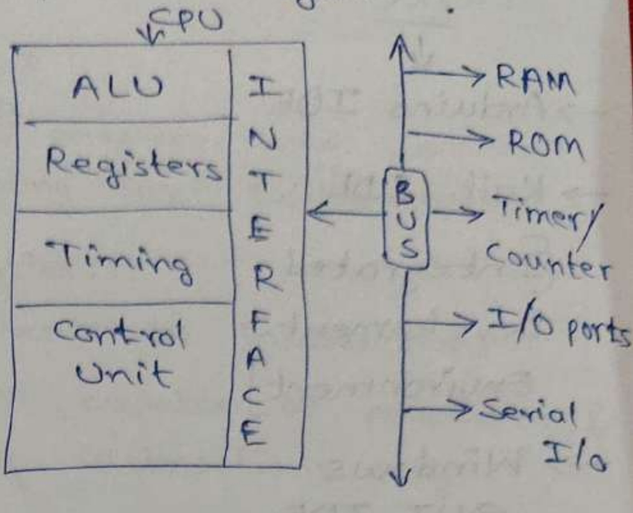
- Microprocessors.



• μc block diagram.

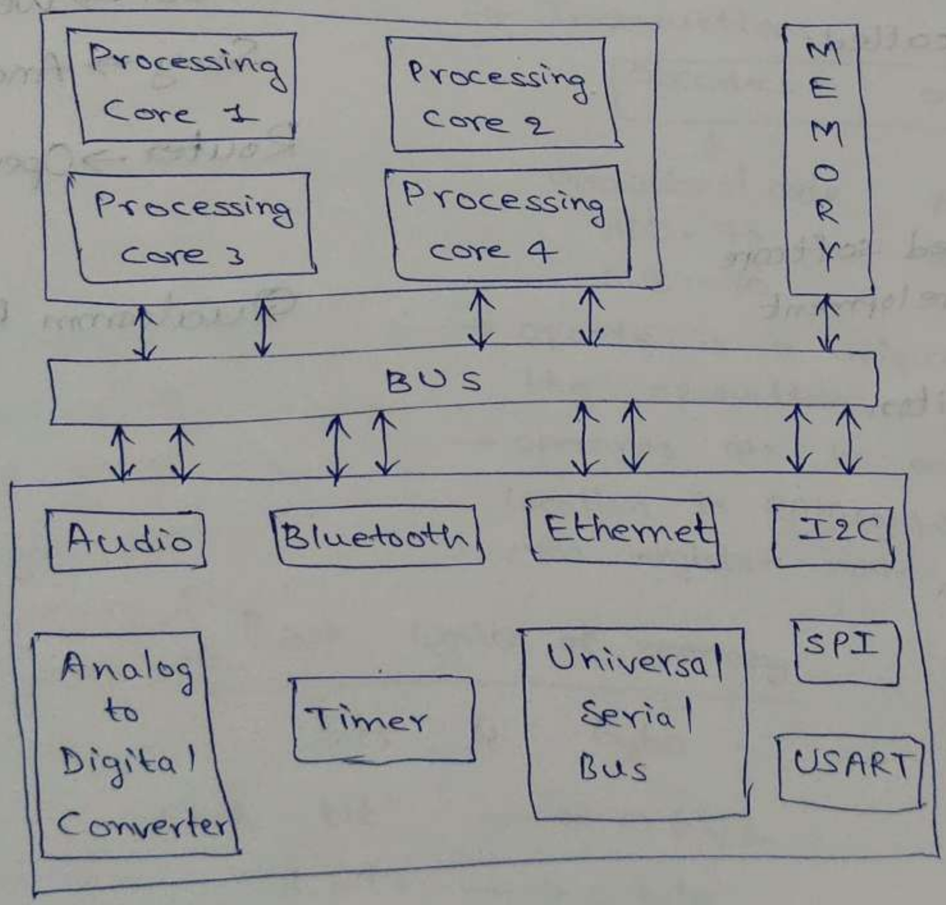


• μp block diagram.



→ SOC's are designed by combining some of best features of μc & μp.

→ Block diagram of system On Chip (SOC):-



μC
↓

→ Arduino IDE

→ Keil IDE
[Integrated Development Environment]

→ Windows GUI IDE

→ All tools needed for embedded software development are integrated into a single software package is called. IDE

Tools → Embedded softcore Development

- → Text Editor
- → Compiler
- → Debugger
- → Loaders/ Linkers

μp
↓

Full fledged OS

- Windouss
- Linux
- Mac

SOC
↓

~~Arduino IDE~~

~~Keil IDE~~

Full fledged OS

→ Embedded Linux
(or)
(Variants, of Linux)

→ Ex:-
→ Android → Linux

Ex:- Smart TV – Samsun
- Tizen (Linux)

LG → WebOS (Linux

Sony → Android (Linux

Router → OpenWrt (Linu

Qualcomm Ride platfo

06-06-2025 Friday

Development Tools

⇛

- Arduino IDE
- Keil IDE
- Text Editor
- Compiler
- Debugger
- Linkers/Loaders

1)→Write : C program are written using Text Editor tools.

2)→ Compilation :

C-statements to instructions

Hardware capable of processing & executing instructions only.

* C statements → Assembly Code → Instructions

High level language → Low level languag

Human Understandable Language → Machine Understandable Language.

→ Instructions

opcodes	operands

opcodes ↓ operational code

ADD – 78

SUB – 70

operands ↓ Address of memory location in RAM.

→ opcode is a unique code that specifies the operation.

→ operands can be address of memory location in RAM(or) Direct Value (or) CPU register names.

⇛

Basic Units of memory

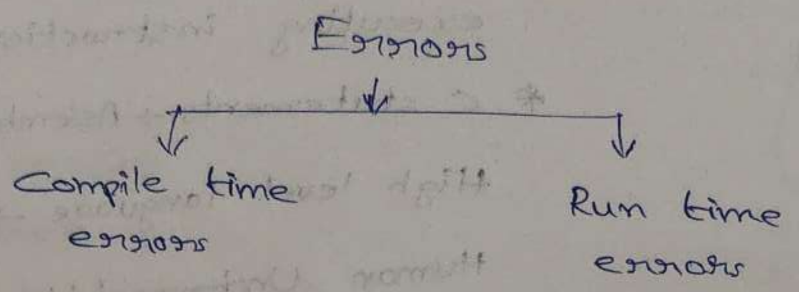
Bits & Bytes

Each bit	⟶ 0 (or) 1
8 bits	⟶ 1 byte
16 bits	⟶ 2 bytes
32 bits	⟶ 4 bytes
64 bits	⟶ 8 bytes

→ RAM

- RAM is divided into bytes.
- Every byte of RAM will have a unique address.
- Addresses are always represent in hexadecimal form.

→ Errors:-



Errors

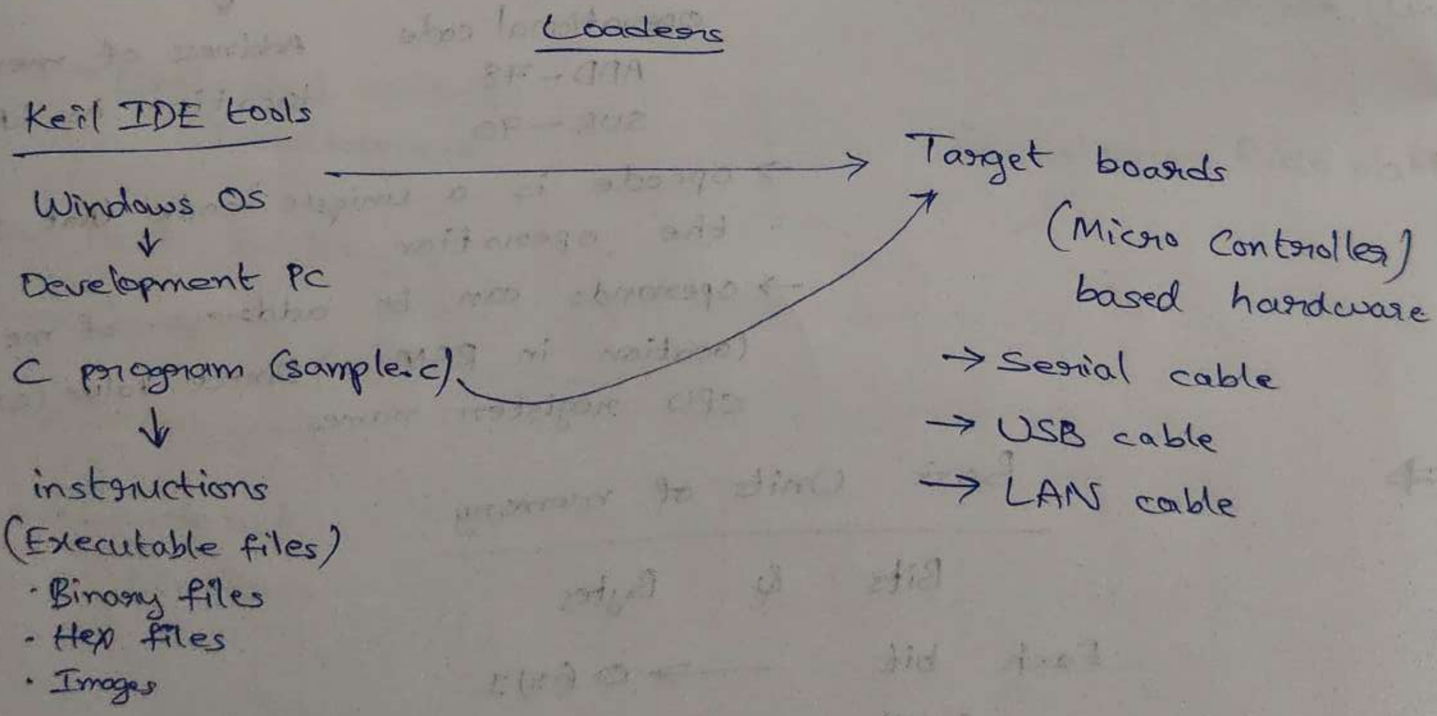
Compile time errors

Run time errors

- To find & resolve compile time errors use compiler

- Debuggers are used to find the run time errors.

→ Loaders:-



Loaders

Keil IDE tools → Target boards (Micro Controller) based hardware

Windows OS
↓
Development PC

C program (sample.c)
↓
instructions (Executable files)

- Binary files
- Hex files
- Images

→ Serial cable

→ USB cable

→ LAN cable

⇻ Linux Environment ——→ Development tools
↓
Ubuntu OS → Text Editor
~~compiler~~
~~debugger~~
~~loader~~

↪ Vi Text Editor (Visual Interface)
VIM (Visual Interface improved)

CLI base tools (Command Line Interface)
(commands) → special application

→ Terminal application → Runs commands

→ Once you open a terminal application, the blinking cursor is called command prompt.

→ To open an existing file `$ vi sample.c`

*→ Terminal application comes pre-installed with every Linux OS.

→ Documentation for Vi text editor (help (or) manual pages)
↳ `$ man vi` ↳ tool name
`$ vi --help`

→ Compiler → gcc (GNU C Compiler)

→ Debugger → gdb (GNU Debugger)

→ Loader → ldd
↳ open an existing file
↳ vi sample.c

* printf() → C standard input/output library function

* ls ——→ to list files in a directory
· white colour are normal files | · Green colour are Executable files.
· Blue colour are directories(or) folders.

* Assembly code files is in .s extension.

* File Permissions (file info)

- File permissions (read / write / execute)
- Username & Groupname. → Login Details
  User
  *****
- Size of file
- Date of creation (Time startup)
- Name of the file.

* Flags (or) Options

```
-a        additional info is called flags (or) options
-l            $ls
-p            $ls -l
                  additional info
                  (flags (or) options)
```

Q) What ~~all~~ flags ~~and~~ or options can be used with the ls command?

Ans) Documentation for ls command / manual paper / help

```
         $ man ls  → gives documentations
manual ←
         $ ls --help
```

⇒ (1) Write a C-program

```
$vi sample.c
```

(2) Compilation

```
         compiler (gcc)
c-statements ──→ instructions                SOC
     ↓                 ↓                       ↳ CPU
sample.c ──────→ executable file → a.out (assembler output)     ↓
                                                          only understands instructions
```

* GCC – GNU Compiler Collection

- It is a free, open-source compiler developed by the GNU project, used to compile C, C++, objective C, and other languages.

- $\underbrace{\texttt{\$gcc sample.c}}_{\text{executable file (a.out)}} \longrightarrow$ C statements
  ↓
  instructions

Q) How to remove a file?

Ans) `$ rm filename`

`$ rm a.out`

Q) How do you get documentation for rm command?

Ans) `$man rm`

(3) Execution / Running a program

$\texttt{\$./}\underbrace{\texttt{a.out}}$ ↳ executable filename

*program execution starts from main() function.

./ } → mentions that executable file is present in the current working directory.

1, Programmers point of view

- Program execution starts from main()
- Every program should have 1 function named main
- main() function.
  ↳ entry point of a program.

2, System / Hardware point of view.

10-06-2025

Tuesday

⇒ Managing Directories or Folders [commands]

* Ctrl+Alt+T → Terminal

* Each directories can contain few more files or sub-directories.

* change to directory → $cd directory-name

* Go back to previous directory → $cd ..

* Once you use Tab it will fill remaining characters afte first character of directory entered.

* Using a single command to jump to 4th directory

  $cd EMBEDDED/LINUX/DRIVERS/I2C

* single command to move back in multiple directory

  . if ~~we~~ use $cd it will move back to main(or) home directory
  root/home/viven.

* $pwd → present working directory
  current working directory.

* $cd/ → root directory will be taken.

* Having separate workspace have seperate directories for multiple users.

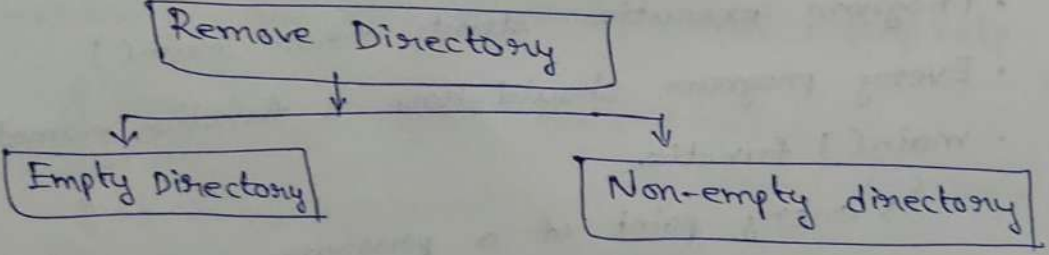
* $mkdr → creates new directory.

  $mkdr <directory-name>
  $man mkdr
  $[illegible]
  $mkdr --help

  Documentation/Help/Manual Pages

* $rm<filename> → removes file

*



- $rmdir directory_name
- $rm -d dir_name

  flags/options

- $rmdir cannot be used to remove non empty directory.
- ~~$rm -d <dir_name>~~ → don't work
- $rm -r dir_name

  ↳ recursively go to every sub directory and remove files.

11-06-2025 Wednesday

→ Vi Text Editor:-

} vi sample.c

Vi – has two operating modes

Q) Explain operating modes of Vi Text Editor?

A) 2 operating modes

- → Insert mode
- → Command mode

1.) Insert mode: Write C-programs/statements

2) Command mode: Administrative tasks
(Save, exit, cut, copy, paste, search)
↳ We can't type/write a C-program

} Ctrl + Shift + '+' → Maximize
Ctrl + ~~Shift~~ '-' → Minimize

- Vi Editor by defaults opens up in command mode.

- Command mode ↓ Insert mode → Special key i

--INSERT--
↳ String appears at bottom left hand side corner

- The new statement is not immediately save to file present in hard disk.
- In PC,
  ↳ Any file is stored in hard disk / secondary memory / Mass Storage Device
  sample.c
  Primary memory → RAM
- In Embedded Device,
  ↳ Files are stored in secondary memory
  - → Internal memory → Flash memory
  - → External memory → SD card, μSD card

primary memory → RAM

Q) How to save new statements to file present in Hard Disk,?

A) Save Operation (Administrative Task)

↑

Command mode

Insert Mode

↓ → Special key Esc

Command Mode

Save Operation → :w<ENTER

write

bottom LHS corner

New statements are saved to file present in hard disk.

- Exit/Quit from Vi Text Editor. (go back to command prompt)
  ↳ :q<ENTER>
- Quit Operation (Administrative Task) { :w + :q
  :wq<ENTER> (save and exit) → :x
  :wq & :x

Q) Can you create an executable file with a different name?

```
$ gcc sample.c
```

a.out

Executable file name

```
$ gcc sample.c -o sample
```

additional instruction flags/options

sample

→ Explanation $ math gcc

```
$./sample
```

12-06-2025 Thursday

⇒ Save & exit → :w + :q (or) :wq (or) :x

Don't save & exit → ~~:x~~ :q!

⇒ Copy & Paste (Administrative Tasks)

↑ command mode

* copy a single line → yy
↳ Yaukee

* pasting a single line → p

* copy two lines → 2yy → no.of lines

* paste two lines → p

* copy four lines → 4yy

* paste four lines → p

*If we are copying more than 2 lines, a notification appears at the bottom LHS. corner.

*If we paste more than 2 lines, a notification appears at the bottom L.H.S. corner.

⇒ copy & paste - word

* copy a single word → yw

* paste a single word → p

* copy two words → 2yw

* paste two words → p

⇒ Delete

* Delete single line → dd
* Delete four lines → 4dd
* If we are deleting more than 2 lines, a notification appears at the bottom L.H.S. corner.

⇒ Search

* forward slash

  /<string name><ENTER>

  ↳ cursor jumps to 1st instance of the string

  bottom L.H.S. corner

* Note:- Search operation is case-sensitive.

  ↳ C is a case-sensitive language.

- Upper case & lower case alphabets are treated differently
- Upper case & lower case alphabets has different ASCII values

Q) How to find next instance of the string (search forward)

Ans) n (next)

* search backward → Shift + n

⇒ Documentation for library function

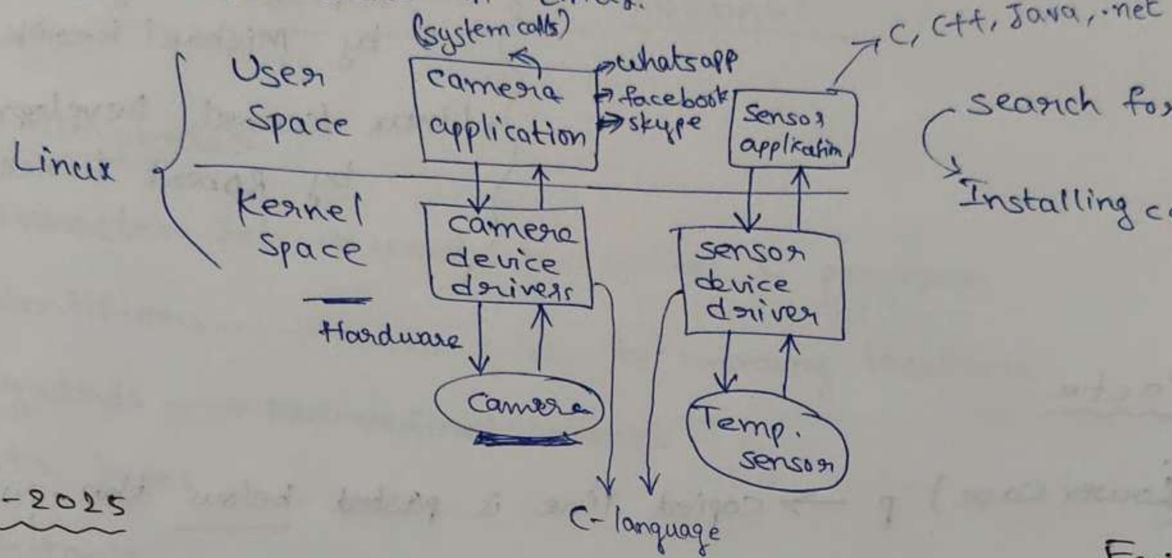
* man printf
* man scanf
* man sleep

→ Library functions

→ System calls (320+)

open()
read()
write()
signal()
sigprocmask()

⇛ Two layers are there in Linux.



compatable
search for drivers
→ Installing compatible driver

13-06-2025

Friday

* Application uses system calls to send request to kernel space drivers.

* Kernel Space [Core OS]
- → Process management subsystem.
- → Memory management subsystem.
- → File management subsystem.
- → Device I/O management subsystem.
- → Network Management subsystem.

Q.) Can application send request directly to device?

Ans) Application can never send request directly to the device.

* Device can never send request directly to the application

Q) Who can initiate an I/O request?

Ans) * Device driver can never initiate I/O request.

* Application can initiate I/O request by using ~~I/O~~ system call.

* Device can also initiate an I/O request with the help of hardware interrupts.

- C language by srivastav
- Linux programming Interface by Michael kerisk
- Linux kernel Development by Robert larice.

⇒ Paste

* (Lower case) p → copied line is pasted below the current line,
* (Upper case) P → copied line is pasted above the current line,

* command mode
  ↓
  insert mode

shifts from command mode to insert mode
→ i

(lower case) o → It creates a new empty line below the current line.

(Upper case) O → It creates a new empty line above the current line.

Q) How to shift cursor to last line?

Ans) Shift cursor to last line → G

* Shift cursor to first line → gg

* Shift cursor to 25th line → :25 <ENTER> → line number

25gg ← → 25G

* To exit from man command documentation we can use q.

* Executable file
(instructions, data)
↳↳
Memory Segments
Memory Sections

16-06-2025

Monday

INTRODUCTION — C LANGUAGE

* Basic Elements:-

1) Character Set → used to write C program.
2) Identifiers ⟶ names given to memory locations.
3) Keywords ⟶ pre-defined words
4) Data types
5) Constants
6) Variables
7) Expressions
8) Statements

* ① Character Set:-

→ There are 4 categories in character set.

i, Alphabets [Upper case & Lower Case] → A, B, C, ... Z
→ a, b, c, ... z

- Every alphabet is associated with unique integer value called as ASCII value.
- ASCII — American Standard Code for Information Interchang
- C is a case-sensitive language, as both upper case & lower case alphabets are treated differently, because they

ASCII values. { A-65 a-97
B-66 b-98
: :

- In order to lookup the ASCII values, type ascii in the terminal. $ascii.

ii, Numerics → [0,1,2,...,9] numeric characters.

- Numeric characters are also associated with ASCII values

0 - 48
1 - 49
: :
9 - 57

iii, Special symbols:-

+, -, /, *, %, ', ", \, &, ...

- Special symbols are also associated with unique integer values called ASCII values.

iv, Backslash characters/Escape sequence:-

- Backslash can be combined with alphabets and also combined with numeric zero

single character {
\n →
\r →
\b →
\t →
\a →
}
\0 → numeric zero

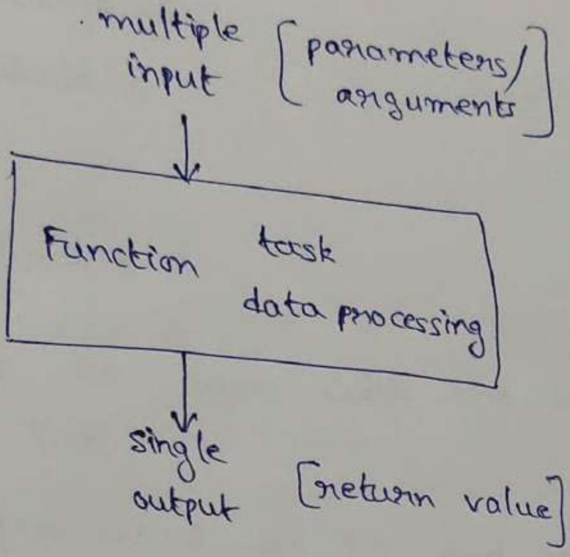
\ + alphabets
\ + numeric zero

- \n is new line character.
- \n, is not treated as two characters, ~~they are~~ it is treated as a single character.
- Most of the backslash characters are used within printf
- \0 - backslash zero is not used within the printf.
- Backslash zero is used for string termination.

* printf is used to display some output on to the screen.

* printf is a C-standard input/output library function.

* Every function is capable of taking multiple inputs and perform task, data processing and gives single output.



* Whatever ~~input~~ information mentioning within the parenthesis, that is basically called as input.

* Backslash characters are used with in printf to change the position of the cursor.

* ~~Bk~~ Backslash characters are non-printable characters.

* \n → shifts cursor to next line starting position.

* \n is represented with a different name as LF (Line Feed). It's ASCII value is 10.

* There are 128 ASCII ~~val~~ characters. (i.e., from 0 to 127).

* The ASCII value for space is 32. Because space is also a character.

* Delete key also has ASCII value.

* Based on printable & non-printable characters the character set is divided in two groups.

* Alphabets, numerics, special symbols — printable characters.

* Backslash characters, escape sequence — non-printable character.

17-06-2025

Tuesday

* \r → Cursor shifts to the starting position in ~~at~~ the same line.

* Ex:-

```
main()
{
    printf ("Hello\r");
    printf ("World");
}
```

Output:- World

* \r → Carriage Return

* \r cannot be seen with the same name in the ASCII character list.

* \r is represented as CR and it's ascii value is 13.

*

printf ("Hello"); → 1 argument / parameter

printf ("%d", x); → 2 arguments

printf ("%d%d%d", x, y, z); → 4 arguments

1st argument ⇒ ~~format specifier~~ control string (%d %c %x)

* \b → Backspace character.

* \b → moves cursor one character backward.

*Ex:-

```
main()
{
    printf("Hello\b");
    printf("World");
}
```

Output:- HellWorld

* \b cannot be seen with the same name in the ASCII character list.

* \b is represented as BS and its ascii value is 8.

* \t → Horizontal Tab

* \t → shifts the cursor one tab space forward.

* Ex:-

```
main()
{
    printf("Hello\t");
    printf("World");
}
```

Output:- Hello $\underbrace{----}_{\rightarrow tab \leftarrow}$ World

* \t cannot be seen with the same name in the ASCII character list.

* \t is represented as HT and its ascii value is 9

Q) Do comments increase the size of the executable file?

Ans: Comments are removed completely during compilation. So, the comments does not increase the size of executable file.

* Compilation stages:- [4 stages]

i, preprocessing

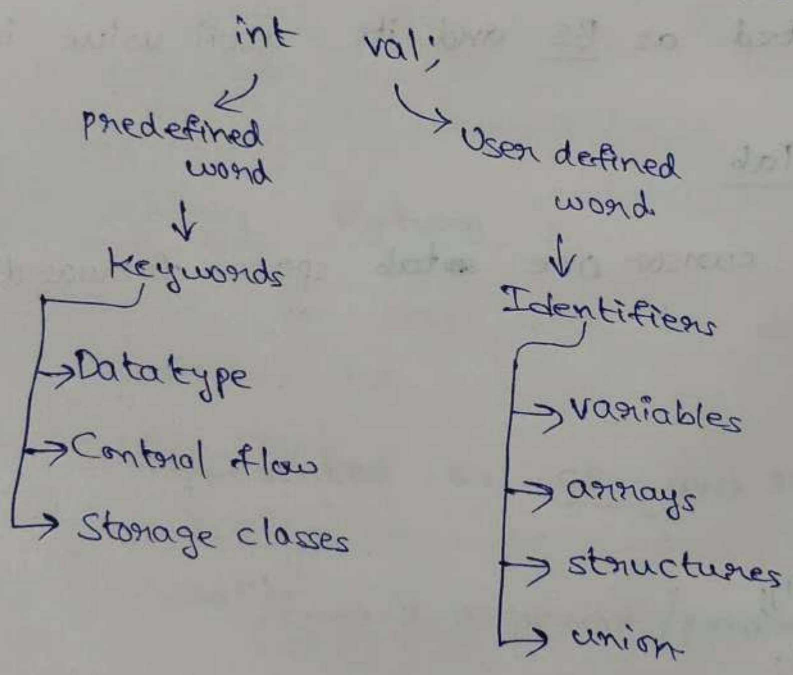
ii, compilation

iii, assembly

iv, linking

- comments are removed in the preprocessing stage.

* If we combine the characters we will get words.



int val;

predefined word

Keywords

Datatype

Control flow

Storage classes

User defined word

Identifiers

variables

arrays

structures

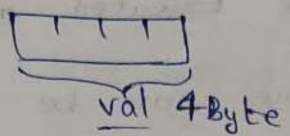
union

18-06-2025 Wednesday

*② Identifiers:- (User defined words)

- User defined words are called as identifiers.
- Names given to memory location.

```
int      val;
↙        ↳ variable
Data type
```



assignment operator
↑
val = 25;

25 is stored in 4 bytes of memory named as Val.

→ System point of view
- creating 4 bytes of memory
- 4 bytes of memory has a name val

→ Rules for naming an Identifier:-

1) Identifiers name can contain alphabets (upper & lower case), numerics, only one special symbol (i.e., _).
   ex:- int val_20;

2. Identifier name can start with alphabets (or) _ (underscore).
   Identifier name ~~can not~~ shoud never start with numerics.

```
ex:- int val; ✓
     int _exp; ✓
     int 5_emp; ✗
```

3.) Identifier names should not use keywords. → 32 keywords

4). C is a case-sensitive language (Upper case & lower case alphabets are treated differently). (has different ascii values).

```
main()
{
  int val;  → generates compilation error.
  int val;    (We cannot have two variables with same
               name in same function)
```

```
main( )
{
   int val;
   int VAL;
 }
```

→ Eventhough they have same name, they are treated as two different variables.

- val, Val, VAL are treated as three different variable.

5) Identifier name can be of <u>any length</u>, but the compiler recognises the <u>first 31 characters</u>.

6) White space characters (or) blank space characters are not allowed within identifier names.

<u>ex</u>:- int net pay ; X
blank space

<u>32-bit</u> compiler
→ gcc (linux)
→ arm-linux-gcc
(cross-compiler)
Recognizes first 31 characters.

<u>16-bit</u> compiler
→ Turbo-c (Windows)
Recognizes first 8 characters.

native compiler
v/s
cross compiler
assignment

→ The same rules will apply for naming a variable, array, structure, and union.

* ③ <u>Keywords</u>:- (Predefined words)

1) Predefined words are called as keywords.

2) Keywords also called as <u>Reserved words</u>.

3) All keywords are mentioned in lower case.

4) There are total 32 keywords in C.

→ Data types:

1) Data type is always associated with variable name.

Data type

↓

Premitive data type (or) Basic data type (or) Built in data types
↓
char
short
int
long
float
double

Non-premitive data type (or) Derived data type (or) User defined data type
↓
Arrays
Structures
Unions

int val;
↓
Variable declaration

→ Before using any variable, declare that variable.

→ By using basic data types, we can create arrays, structures, and unions.

Q) What does a data type specify?

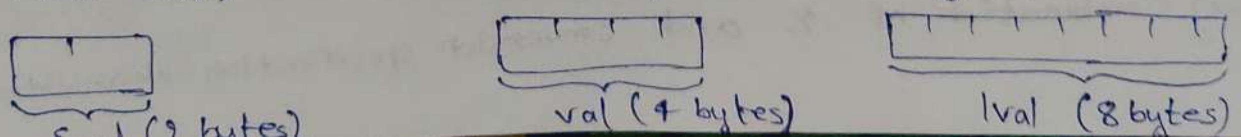
A) 1, How many bytes the corresponding variable occupies?
2, What type of data stored in the corresponding variable?
3, Range of values that can be stored in the corresponding variables.

→ char ch;

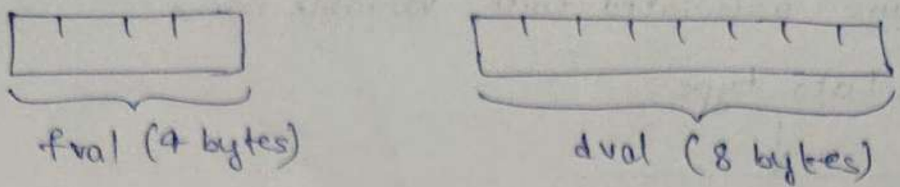
ch (1 byte)

You are creating a 1 byte of memory, this 1 byte of memory has the name ch.

→ Short Sval; →int val; →long lval;



→ float fval; → double dval;



short
int
long
} integer values

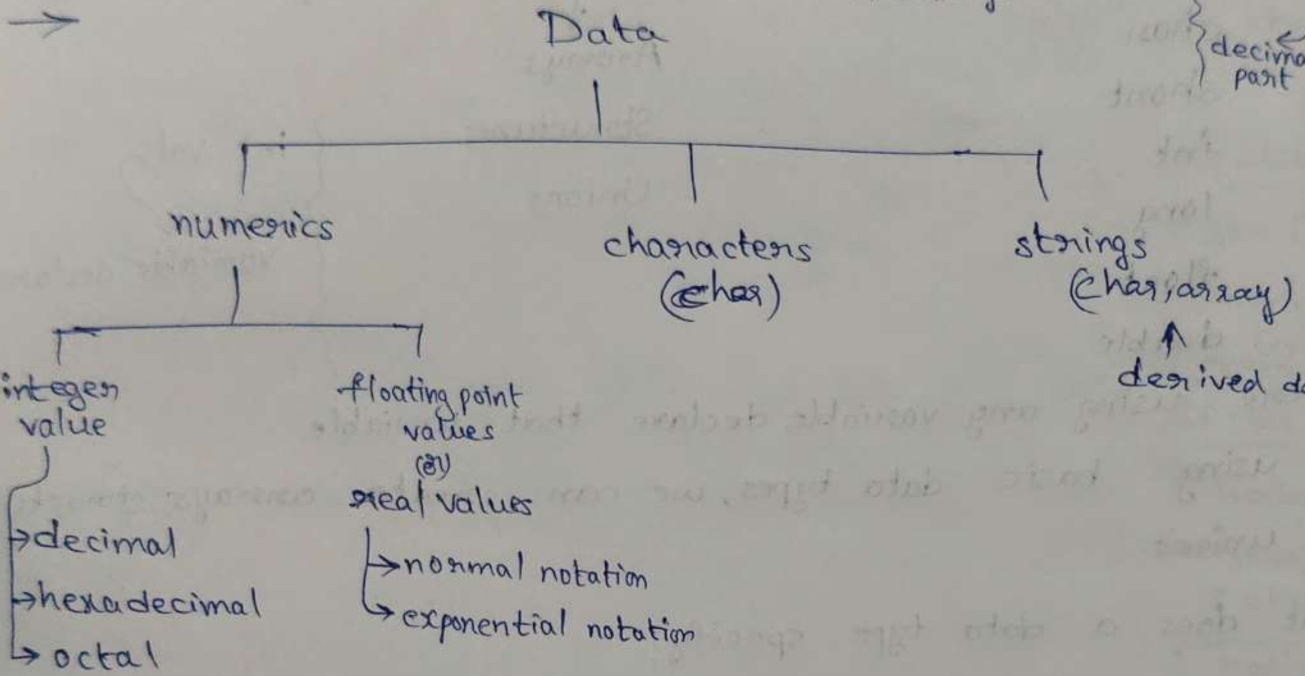
long
double } floating value

→ We don't have any data type for strings, we have to use derived data types - character, array.

5.9 feet

5.9
decimal part
decimal point
fraction part

→

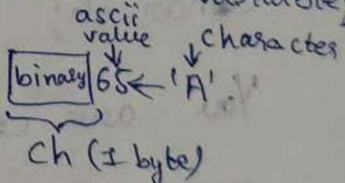


19-06-2025 Thursday

⇛ Range of values → character variable

Q.) How do you store a character in a character variable?

Ans) Char ch = 'A'; ~~char ch = A~~;



- A character should enclose within single quotes.

Q) What is format specifier?

Ans) Combination of % and conversion specification characters.

* printf("%c", ch); → A

(binary data) → ascii value 65 → ascii character A

- printf & scanf functions uses these format specifiers to specify the size and type of data.

%d
%x
%o
%c
%f
%e
%i
%g

} conversion specification character.

- The format specifier %c specifies the contents of the variable which has binary data converts to ascii value then converts into ascii character.

- ASCII value is a integer value.

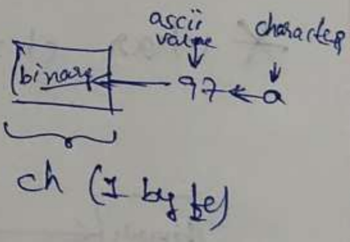
* printf("%d", ch); → 65

binary data → ascii value 65

- Format specifier %d specifies the contents of the variable which has binary data convert to ascii value.

* char ch;

ch = 'a'; a is stored in 1 byte of memory named as ch.

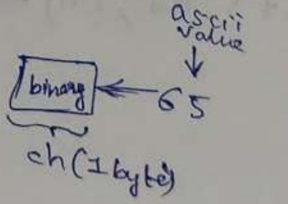


printf("%c", ch); → a

printf ("%d", ch); → 97

* We can store integer values within a character variable

char ch = 65;



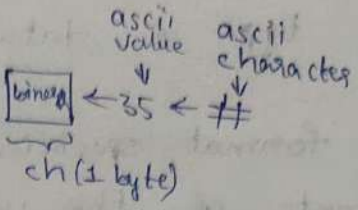
NOTE:- We can store only specific range of values-(sign qualifier) printf("%c", ch); ⟶ A

printf("%d", ch); ⟶ 65

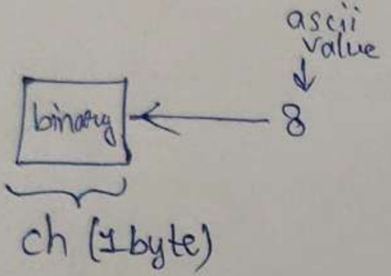
* char ch='#';

printf("%c", ch); ⟶ #

printf("%d", ch); ⟶ 35



* char ch = 8;

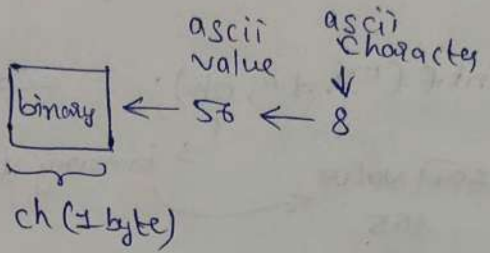


printf("%c", ch); ⟶ Don't see any o/p

(BS) ascii character ← ascii value(8) ← (binary value)

printf("%d", ch); ⟶ 8

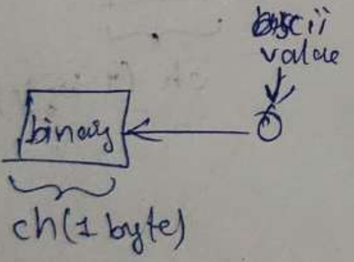
* char ch='8';
↳ numeric character



printf("%c", ch); ⟶ 8

printf("%d", ch); ⟶ 56

* char ch=0;

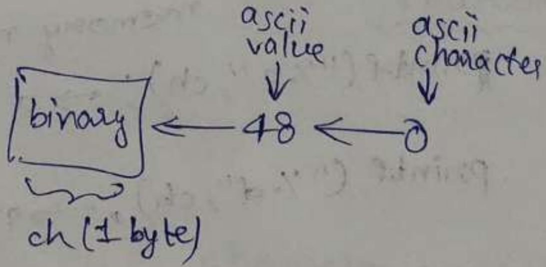


printf("%c", ch) ⟶ No o/p is displayed

(\0) ascii character ← ascii value(0) ← (binary value)

printf("%d", ch) ⟶ 0

char ch='0';



printf("%c", ch) ⟶ 0

printf("%d", ch) ⟶ 48

char ch='0';
char ch = '\0';

* char ch = '\0';

```
printf("%c", ch);  → No o/p displayed      (ASCII value = NULL)
printf("%d", ch);  → 0
```

20-06-2025

Friday

⇒ String :-

Multiple characters
Sequence of characters } enclosed within " " (double quotations)
Group of characters

- "Linux" →

'L'	'i'	'n'	'u'	'x'	'\0'

5 + 1 (null character)
6 bytes

compiler stores null character at the end of the string.

- A null character is always stored at the end of the string.

⇒ Range of values that can be stored in a character variable.

Q) What is the range of values that can be stored in a character variable?

Ans:- 1) Range Depends on sign qualifier
→ signed
→ unsigned

Q) If you ~~the~~ don't apply any sign qualifier, which one applies by default?

Ans) ~~Signed qualifier~~ Signed.

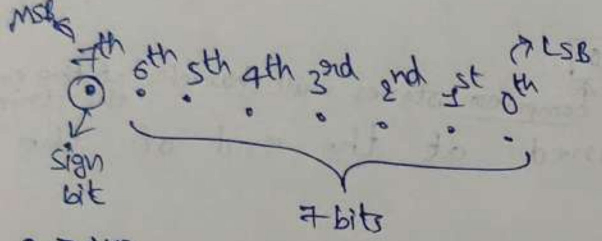
⇒ Signed char ch;

1) Can store both positive & negative values.

2) Most Significant ~~Byte~~ Bit (MSB) is treated as sign bit.

- In sign bit, every bit is capable of holding either 0 (or) 1.
- Sign bit :- 0 → +ve
  1 → -ve

char → 1 byte
↓
8 bits



0 → +ve
1 → -ve

$2^7 \rightarrow 128$
↳ possible values that can be stored in 7 bits.

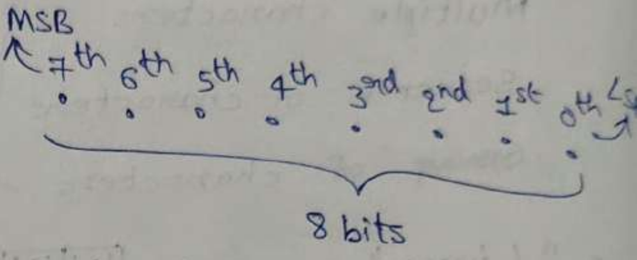
3) -128 ↔ 0 ↔ 127

Unsigned char ch;

1) Can store only positive values.

2) MSB is not treated as signed.

3) 0 ↔ 255



$2^8 \rightarrow 256$
(0-255)

char ch=127;
ch=ch+1;
↳ 127+1=128
printf("%d", ch);
↳ -128

⇒ Signed character variable:-

1) Check if the result is within the range.

2) Result 128 is not within the range.
-128 ↔ 0 ↔ 127

3) Result has exceeded the maximum value in the range by:

4) Once you exceeded the maximum value in the range wrap around occurs.

5) Once the wrap around occurs counting again starts from minimum value in the range.
-128

```
* char ch=127;
  ch=ch+3;
     ↳ 127+3 (=130)
  printf("%d", ch);
     ↳ -126
```

∵ 130 is not within the range, so wrap around occurs

```
* char ch=0;
  ch = ch-1;
     ↳ 0-1=-1
  printf("%d",ch);
     ↳ -1
```

∵ -1 is within the range, so wrap around doesn't occurs.

```
* char ch=-128;
  ch = ch+1;
     ↳ -128+1 =-127
  printf("%d",ch);
```

∵ -127 is within the range, so wrap around doesn't occurs.

```
* char ch=-128;
  ch=ch-1;
     ↳ -128-1=-129
  printf("%d",ch);
     ↳ 127.
```

1). Result is not within the range.

2) Result has gone below the minimum value in the range by 1.

3) Once the result goes below the minimum value in the range wrap around occurs.

4) Once wrap around occurs counting again starts from maximum value in the range.

⇒ Unsigned character variable [0 ⟷ 255]

* unsigned char ch=255;

```
    ch=ch+1;
        ↪255+1=256
printf ("%d", ch);
        ↪0
```

1) 256 is not within the range.
2) Result has exceeded the max. value in the range by 1
3) Once result exceeds the max. value, wrap around occurs
4) Once wrap around occurs, counting again starts from min. value.

* unsigned char ch;

```
ch=0;
ch=ch-1;
    ↪0-1=-1
printf("%d", ch);
    ↪255
```

1) -1 is not within the range.
2) Result has gone below the min. value in the range
3) Once result goes below the min. value, wrap around occurs.
4) Once wrap around occurs, counting again starts from max. value.

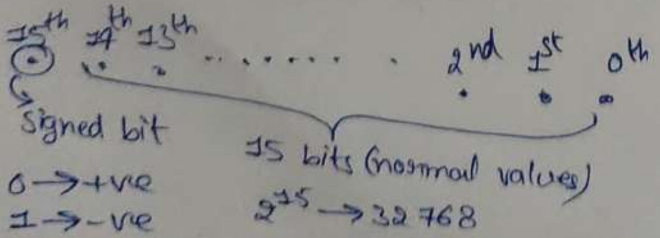
23-06-2025

Monday

⇒ Short Variable:-

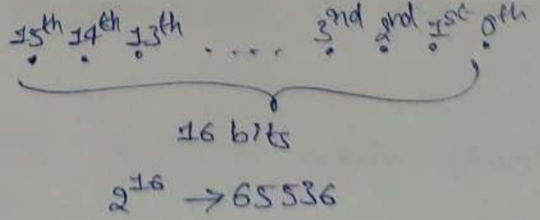
Signed short sval;

1) Can store both positive & negative values.
2) MSB is treated as signed bit
   ↑ Most significant Bit.
3)



Unsigned short sval;

1) Can store only positive values
2) MSB is not treated as signed bit.
3)



Values range from
-32768 ↔ 0 ↔ +32767

Values range from
0 ↔ 65535

⇒ Signed short variable [-32768 ↔ 0 ↔ +32767]

* Short sval = 32767;
sval = sval+1;
→ 32767+1 = 32768
printf("%d", sval);
→ -32768

1). Value is not within the range
2) Result has exceeded the max. value in the range by 1.
3) Wrap around occurs → counting starts from min. value.

* Short sval = 32767;
sval = sval + 3;
→ 32767+3 = 32770
printf("%d", sval);
→ -32766

→ During additions & subtractions we can see the wrap around technique is being applied.

* Short sval = -32768;
sval = sval + 1;
→ -32768+1 = -32767
printf("%d", sval);
→ -32767

1) Result is within the range.
2) So, no wrap around is applied

* short sval = -32768;
sval = sval - 1;
→ -32768-1 = -32769
printf("%d", sval);
→ 32767.

1) Result is not within the range.
2) Result has gone below the min. value in the range by 1
3) Wrap around occurs → counting starts from max. value

⇒ Unsigned short variable [0 ↔ 65535]

```
* unsigned short sval = 65535;
  sval = sval + 1;
        ↪ 65535+1 → 65536
  printf("%d", sval);
        ↪ 0
```

1) Value is not within the range.
2) Result has exceeded the max value by 1.
3) Once result exceeds the max value, wrap around occurs.
4) Once, wrap around occurs, counting again starts from min. value

```
* unsigned short val = 32767;
  val = val + 1;
        ↪ 32767+1 = 32768
  printf("%d", val);
        ↪ 32768
```

1) Result is within the range
2) No wrap around technique is applied

```
* unsigned short val;
  val = 65535;
  val = val + 3;
        ↪ 65535+3 = 65538
  printf("%d", val);
        ↪ 2
```

```
* unsigned short val;
  val = 0;
  val = val - 1;
        ↪ 0-1 = -1
  printf("%d", val);
        ↪ 65535
```

1) Value is not within the range
2) Result goes below the min. value by 1.
3) Once result goes below the min. value, wrap around occurs.
4) Once wrap around occurs, counting again starts from the max. value

⇒ Integer variable;

Signed int val

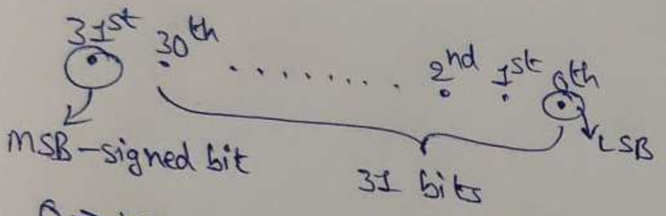
1) Can store both positive & negative values

2) MSB is treated as the signed bit.

3) -2147483648 ↔ 0 ↔ +2147483647

-200 crore ↔ 0 ↔ +200 crore

-2 billion ↔ 0 ↔ +2 billion



0 → +ve

1 → -ve

$2^{31} = 2147483648$

Unsigned int val

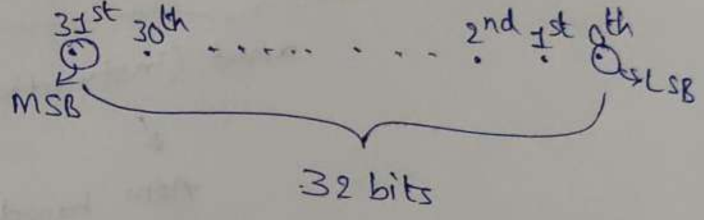
1) Can store only positive values.

2) MSB is not treated as the signed bit

3) 0 ⟷ 4294967296

0 ↔ 400 crore

0 ↔ 4 billion



$2^{32} = 4294967296$

⇒ int val;

* 4 bytes ; signed int val → Range
unsigned int val → Range

* type of values → integer value (dec, hexa, octal)

* 16-bit compiler
int → 2 bytes (Turbo C Windows)

* 32-bit compiler
int → 4 bytes (gcc compiler)

→ The instruction set for Intel architecture & ARM architectu
is completely different.

→ Development always done in Desktops / Laptops (i.e, intel)

→ intel
gcc sample.c

a.out (instructions)

↙ intel ~~↘~~ ARM

intel
↑
→ arm_linux_gcc sample.c

a.out (instructions)

↙

ARM board