16/6/25 Introduction – C language

Basic elements:
1. Character set : are used to write C programs.
2. identifiers : Names given to memory location.
3. keywords : predefined words.
4. Data types :
5. Constants
6. Variables
7. Expressions
8. Statements.

1. Character set : are categorised in 4 types.
   i) Alphabets (uppercase & lower case) (A,B,C.....,Z ; a,b,c.....z)
   Every alphabet is associated with unique integer value called ASCII value.
   ASCII – American Standard Code information Interchange.

A – 65	a – 97
B – 66	b – 98
:	:
Z – 90	z – 122

   * C is a case sensitive language. as both upper case & lower case alphabets treated as different ASCII values.

   ii) numerics (0,1,2,.... 9)
   => numeric characters also associated with ASCII values.

0 – 48
1 – 49
:
9 – 57

   iii) Special Symbols (*, +, –, &, ^, ...)
   -> Special Symbols are associated with unique integer values is called as ASCII value.
   iv) Back Slash Characters / Escape sequence :

\n →
\r →
\b →
\t →
\a →
\0 →

\ + alphabet
\ + numeric zero
These are not treated as two characters, it treated single character.

* Most of the back slash characters are used with in printf.
* \0 is not used with in printf. is used for "String termination."
* printf() is used to display some output.
* The parameters are provided with in "parathesis."

(parameters or arguments) multiple input → function task data processing → single output (return value)

printf("Info");
input.

* we can provide multiple inputs using comma(,)
* \ characters are used with in printf because to change or alter cursor position. Expect \0.

printf("Hello\n");
Printable characters. Non printable.

⇒ \n → Shift cursor to "next line starting position"
⇒ \n is represented different Name is 'LF' - line feed → is ASCII - 10
⇒ how many characters set → Total 128 Character ASCII Values (0-127)

\n - new line character
line feed - LF.

~~for~~ space → 32 ASCII vale.

for Delete
DEL - 127 ASCII.

1. Alphabets
2. numerics
3. Special symbols
} printable Characters.

4. back slash / Escape sequence } Non printable characters.

\r ⇒ cursor shifts to "starting position" at the same line.

main()
{
printf("Hello\r");
printf("world");
}

output: world.

world
Hello\r

⇒ \r - carriage return.
⇒ \r can't be ~~same~~ with the same name. ASCII character list.
It seen / represented with "CR" - ASCII "13"

⇒ printf("Hello"); → 1 argument/parameter.
control string ← may or may not format specifier.

⇒ printf("%d", x); → 2 argument/parameter.
format specifier.

⇒ printf("%d %d %d", x, y, z); → 4 arguments/parameters.

⇒ \b – Backspace character., BS – ASCII value – 8

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

output : Helloworld.

⇒ \b → It moves cursor one position backward. direction.

⇒ \t → It shifts cursor one forward direction

"tab space"

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

output :
"TAB space"
Hello ----→ world.
one TAB space = 4 spaces

⇒ \t ⇒ Horizontal TAB ; HT – 9

---

* "Comments are removed during Compilation."
Completly

Compilation stages ← 4 stages.

1. Preprocessing (comments are removed in this stage).
2. Compilation
3. Assembly
4. Linking

---

int val;

- int → Predefined words → Keywords / Reserve words
- val → user defined words → identifiers.

Keywords
- Data type
- Control flow
- Storage class

identifiers.
- Variables
- Array
- Structure
- union.

Escape Sequence	Meaning	ASCII Value	purpose
\0	null character (NUL)	0	→ used for termination of character string
\a	alert (BEL)	7	→ produces an audible or visible alert.
\b	back space (BS)	8	→ move cursor to the previous position of the current line.
\t	Horizontal tab (HT)	9	→ move cursor to the next horizontal tab position
\n	new line (LF)	10	→ to the beginning of the next line
~~\f~~	~~form feed~~		
\v	vertical tab (VT)	11	→ to next vertical tab position
\f	form feed (FF)	12	→ to the initial position of the next logical page.
\r	carriage return (CR)	13	→ to beginning of the current line.
\\	backslash	92	→ presents a character with backslash (\)

18/6/25

Identifiers.

1. user defined word called as identifier.
2. Names given to memory location.
   System point of view: int val;
   * creating 4 Bytes of memory
   * 4 Byte of memory has a name val

assignment operator.

Val = 25

↓ is stored in 4 Bytes of memory named as Val.

Rules for naming an identifier.

1. identifier names can contain alphabets (upper, lower), numerics, only one special symbol (_)
2. identifier names can start with alphabets (or) underscore (_).
   * identifier name "can't start with numerics"

   int val; ✓
   int _temp; ✓
   int 5_temp; ✗
3. identifier names should not use keywords. (32)
4. C is case sensitive language (upper & lower case alphabets treated as different) (because ASCII values)

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

It generates compilation error, because (we can't have two variables with same name in same functions)

(If they same name they treated as two different variables")

5. identifier names can be of any length but the compiler recognizes the '31' (first) characters.

=) for windows the compiler name is "Turbo-C" (windows) (16-bit Compiler) (8 characters recognized)

=) for linux the compiler name is 'gcc' (GNU C Compiler) (32-bit-Compiler) (31 characters recognized)

native compiler vs cross compiler → Assignment.

6. white space (or) blank space not allowed with in identifier Names.

* The same rules will apply for Naming a Variable, Array, Structure, and union.

Keywords & (Predefined words)

* Keywords also called as "reserved words".
* 32 keywords.
* All these keywords are mention in lower case.
* catagories → 1. Data types 2. Control flow 3. Storage class.

1. Data types &
* Data type is always associated with Variable name.
* Data types are two types.

int value; → Variable declaration

Data types
- primitive (or) (Built-in data types) — Basic data types (Char, short, int, long, float, double)
- Non primitive (or) derived data type (or) user defined data type (Arrays, Structure, union)

* By using Basic Data types we created Array, Structure, and Unions.

Q. What does a data type specified?

A. ⇒ How many bytes the corresponding variable occupies.
⇒ what type of data is stored in the corresponding Variable.
⇒ Range of values that can be stored in the corresponding Variable.

* Before using any variable, declare that name.

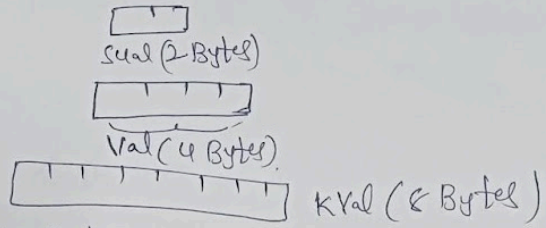
⇒ Char ch;

* we are creating 1 Byte of memory as name with ch.

⇒ Short Sval;

⇒ int Val;

⇒ long KVal;

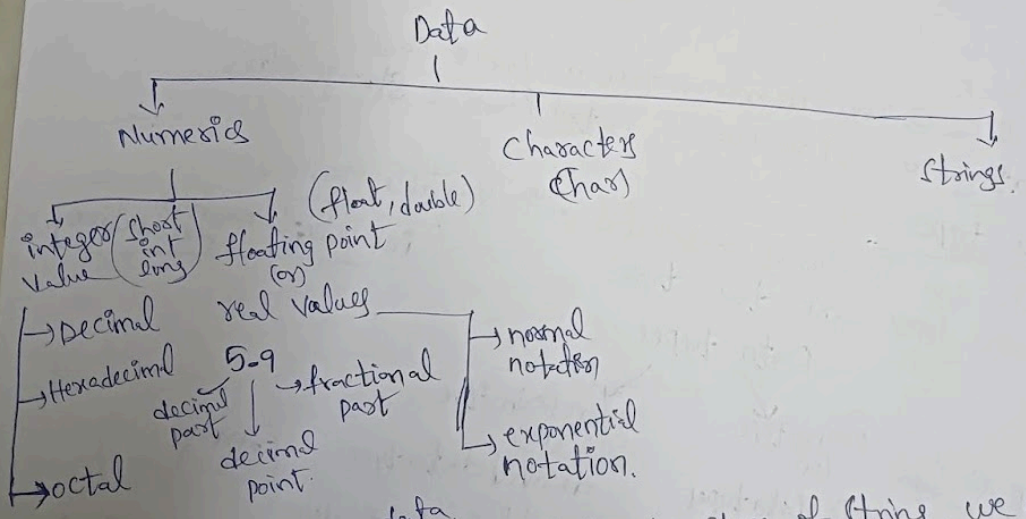


⇒ float fval; 4 Bytes

⇒ double dval; 8 Bytes

Short, int, long } integer values.

~~long~~ float, double } floating values.



* we don't have any type of ~~string~~ data. to store of string, we used character array.

Char ch = {'0' - 32}
non printable characters.

printf("%c", ch);
Don't see output in display.

Char ch = 0;
printf("%c", ch); → No output in display.
printf("%d", ch); ⇒ 0

Char ch = '0';
printf("%c", ch); ⇒ 0
printf("%d", ch); ⇒ 48

Char ch = '\0';
printf("%c", ch); ⇒ No output in display.
printf("%d", ch); ⇒ 0.

20/6/20

String

multiple characters sequence / characters group of characters } enclosed with in " ".

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

Q. Who store '\0' at end of the string? (A) Compiler.

Q. Range of values that can be stored in character variable?

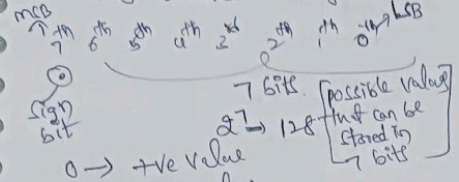
1. Depends of sign Qualifier.
   - signed
   - unsigned.

Q. If you don't apply any sign Qualifier what apply for it?

(A) Signed Qualifier.

Signed Char ch

1. Can store both +ve, -ve values
2. MSB treated as sign bit.



0 → +ve value
1 → -ve value

3. Range.
   -128 to 0 to 127.

unsigned. Char ch.

1. Can store only +ve values.
2. MSB is not treated as signed bit.
3. Range. 8 bits

$2^8 \rightarrow 256$.

0 to 255

for signed characters

1. Check if the result with in Range.
2. Result 128 with in Range
3. Result has exceed maximum Range by 1.

```
Char ch = 127;
ch = ch + 1;
printf("%d", ch);
```
O/P = -128

for ch = ch+3
then o/p = -126.

*once exceed the maximum value in the Range, then "wrap around" occurs.
5. once the wrap around occurs, then counting again start from minimum value in the Range. i.e, (-128).

for unsigned characters (0-255)

1. Check if the result with in the Range
2. If exceeds by 1, then wrap around occurs.
3. once wrap around occur then counting starts from minimum value. (0).

```
unsigned char ch = 255
ch = ch + 1
printf("%d", ch);
```
O/P = 0

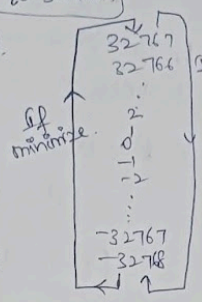
2316125

Q. What is the Range of values for short.

signed short sual;	unsigned short sual;
1. Both +ve, -ve,	1. only +ve values.
2. MSB treated as Sign Bit	2. MSB not treated as signed bit.
15th 14th ..... 1st 0th ↓ 0 → +ve value 1 → -ve value.	3. Range. 2 Bytes - 16 Bits (15th .. 0th)
3. Range $2^{15} \rightarrow 32,768$ -32768 to 0 to 32767	$2^{16} \rightarrow 65,536$ 0 to 65,535

for signed short: (-32768 to 32767.)

* short sval = 32767.
  sval += 1;
  printf("%d", sval);
  o/p :- -32768. ✓
* short sval = -32768
  sval -= 1;
  printf("%d", sval);
  o/p :- 32767. ✓

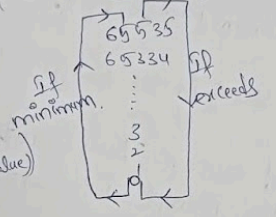


steps
* when check the value with in the 'Range'
* once exceeds (or) minimize by some number then 'wrap around' occurs.
* once wrap around occurs then the count start from
  → minimum value for exceeds (-32768.)
  → maximum value for minimizes (32767)

* During Additions (or) Subtractions we can see 'wrap around' technique being apply.
* Result with in the Range then "wrap around technique" is not applied.

for unsigned short val; (0 to 65535.)

* unsigned short val;
  val = 65535;
  val += 1;
  printf("%d", val);
  o/p :- 0 ✓ (count start from minimum value)
* unsigned short val;
  val = 0;
  val -= 1;
  printf("%d", val);
  o/p :- 65535 (Count start from maximum value).



Signed int val; (4 Bytes) (32 bit)

1. Can store Both +ve, -ve values.
2. MSB is treated as Signed bit.

31st 30th 2nd, 1st, 0th
↑ MSB Signed bit (31 bit)
$2^{31} \rightarrow 2,147,483,648$

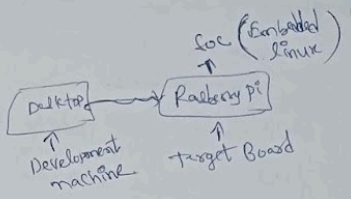
Range :- -2 billion to 0 to 2 billion.
-200 crore to 0 to 200 crore.

Unsigned int val:

1. Can store only +ve values.
2. MSB not treated as signed bit.
   31st 30th 2nd, 1st, 0th
   $2^{31} \Rightarrow 4,294,967,296$
3. Range 0 to 4294967296
   0 to 4 billion.
   0 to 400 crore.

int Val;

* 4 bytes → signed int → Range
  → unsigned int → Range
* type of value → integer value
* 16-bit Compiler (Turbo C windows)
  int → 2 Bytes
* 32-bit Compiler (gcc Linux)
  int → 4 Bytes



Q. ~~Com~~ ~~work~~ Instruction set for Intel and Instruction set for Arm architecture are completely different.

Q. Native Compiler?

The ~~Compiler~~ Development works on in Desktop or laptops then Compiler works in same system then it is called native compiler.

⇒ gcc sample.c

Q. Cross Compiler?

24/6/25

Variables;

* variables are Names given to memory location.
* Variables are Names that can be used to store a value.
* Can store different values but one at a time.
* Value present in the variable changes throughout your program execution
* Before using a variable, declare that variable. int Val;
* A variable is always is allocated with a data type. and decides what values to be stored in that variable.

Val → variable
25 → constant

Val = 25;

* Compiler rules clearly states that we can't have a constant at left hand side of assignment operator. 25 = Val; ✗ Compiler error.
*

Val = x; ✓ valid
x = Val; ✓ variable

Val = x + 10; ✓ valid
Expression

Expression → * Combination of Variable, Constants, operator;

* Variable and constants can be Combined using operator to form Expressions.

* Variables and constants are ~~also~~ called as "operands".

val = x + 10; → operands, operator

⇒ we evaluate the expression we get the Result, the Result assign to the variable name of val.

* we can not have expression on the LHS of assignment operator;

⇒ x + 10 = val ✗ ∴ compilation error.

* val = add (x, y); → functions invocation.

Return value ← input

⇒ Every function is capable of ~~writing~~ returning a value, and these Return value is substituted to function invocation, then Result assigned to variable.

⇒ add(x, y) = val; ✗ ∴ compilation error.

Q. ~~int x;~~

x = printf("Linux");

printf("%d", x);

→ Return the no. of characters are displayed on computer screen.

O/p → Linux5

→ backslash characters are also counted.

Q. int x;

x = printf("Hello\n");

printf("%d", x);

O/p → Linux

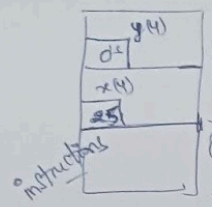
6

file stored.

pc/Desktop
Laptop
↓
Hard disk

Embedded systems
Smartphone / Smart TV
↓
Internal memory → flash
External memory → SD cards, micro SD cards

Memory segments / memory sections / memory layout



a.out

```
int x=25;  → global initialized variable (Data segment)
int y;     → global uninitialized variable.
main()
{
  int z;
}
```

* variables are declared outside of the main function is called global variables.
* Assigning values during declaration is called "initialization".
* Memory for global initialized variables present at Data segement.
* Memory for global uninitialized variable present at BSS segment
  BSS ⇒ Block started after symbol.
* gedit → to see content of .c files.
* we can't used text editor to view contents of executable files.
* we have special tool ⇒ Objdump ✓ readelf

```
$ objdump -s a.out
```

text Editors
GUI → gedit
CLI → vi/vim