Computer Generation

① First gen :- (1940 - 1956) → storage

- Used Vacuum Tubes.

Ex ENIAL, EDVAC

Primary Storage :-

- Magnetic drums.
- Delay line memory.

Secondary storage :-

- Punch Cards
- Magnetic Tapes

② Second gen :- (1956 - 1963)

- Used Transistors

- Ex IBM 1401, IB 7094 → Used - Magnetic disks.

③ Third gen :- (1964 - 1971) → Used - Semiconductor Memory

- Used Integrated circuits (IC)

- IBM 360, PDP-8

④ Fourth gen :- (1971 - Present) → RAM, ROM (Primary)

- Used Microprocessors.

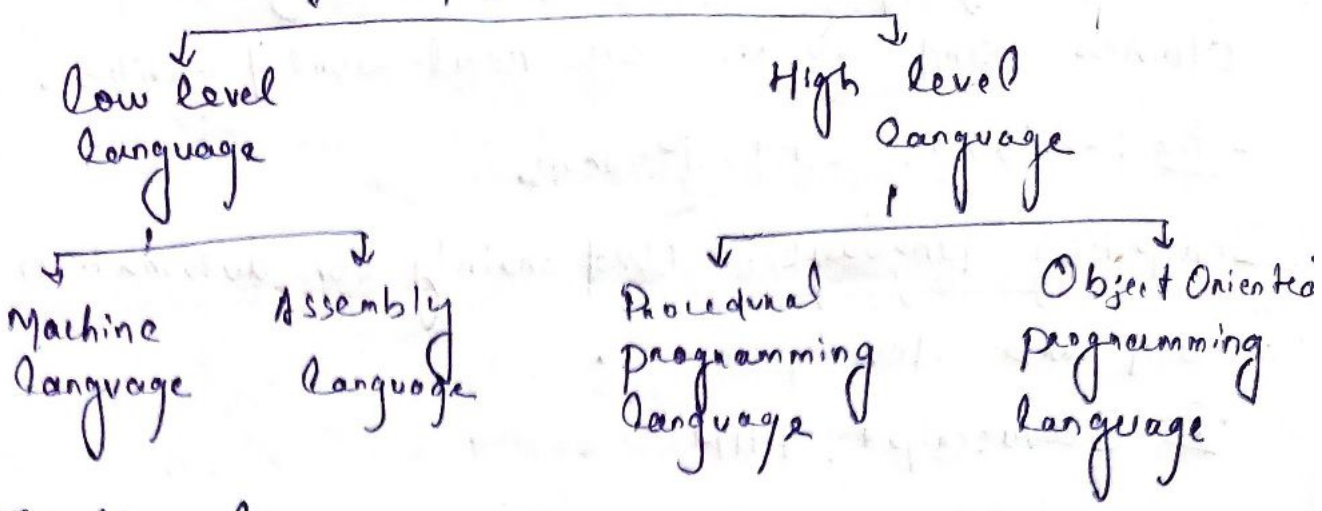
- Ex IBM PC, Apple Macintosh.

→ HDD, SSD, CD/DVD (Secondary)

⑤ Fifth gen :- (Present & future) → SSD, Cloud storage.

- Used Artificial intelligence (AI)

Programming Language



- **Machine language** :- Binary language, it works on binary digits.
- **Assembly language** :- Otherwise known as **Mid level language**.
  - Replace 0 & 1 with english instruction Called pneumonics.
    Ex - MOV, add, sub etc.
  - → "**Assembler**" is used to Convert assembly language into Machine level language.
- **Procedural programming language** :- (Imperative P.lang.)
  - → These language follow a step-by-steps procedure to solve a problem.
  - → Ex - C, pascal.
  - → **Database-oriented language** :- used to store, retrieve and manage data.
    Ex SQL, MongoDB.
  - → **fourth gen language** :- (4GL) → very close to human language and easy to use.
    Ex MATLAB, SAS etc.

Object Oriented programming language :-

- These languages are based on objects and classes and focus on real-world entities.
- Ex :- java, C++, Python.

Scripting language :- Used mainly for automation and web development.
- Ex Javascript, PHP

* A high level language is human Understandable but Computer Can't Understand.
* To make Computer Understandable we have to translate HLL into MLL. So Compiler & interpreter used as translators.
* Compiler Check all errors at a time and Compile whole program at a time.
* Interpreter translate line by line and also check error line by line.

Algorithms :- Step by step process to execute a problem.

Step-1 - Start / Begin

Step-2 -

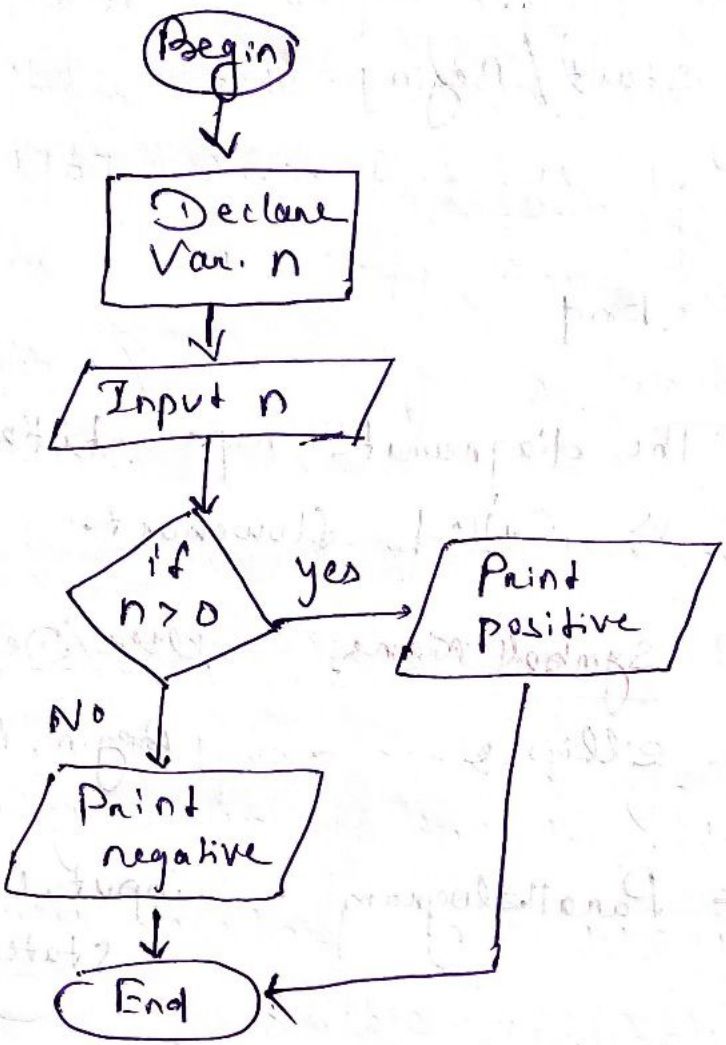
⋮

Step-n - End

flowchart :- The diagrematic representation of an algorithm is called flowchart.

	Symbol	Symbol Name	Uses / Description
1.		- ellipse	- Begin, End
2.		- Parallelogram	- input, Output statement
3.		- Rectangle	- Processing, Constants
4.		- Diamond	- Conditions
5.		- Arrow	- flow of direction
6.		- On page Connector	- To Continue the flowchart in the same page
7.		- Off page Connector	- To Continue the flowchart in the next page

Ex - Check wheather a number is +ve or -ve.



Pseudocode :-

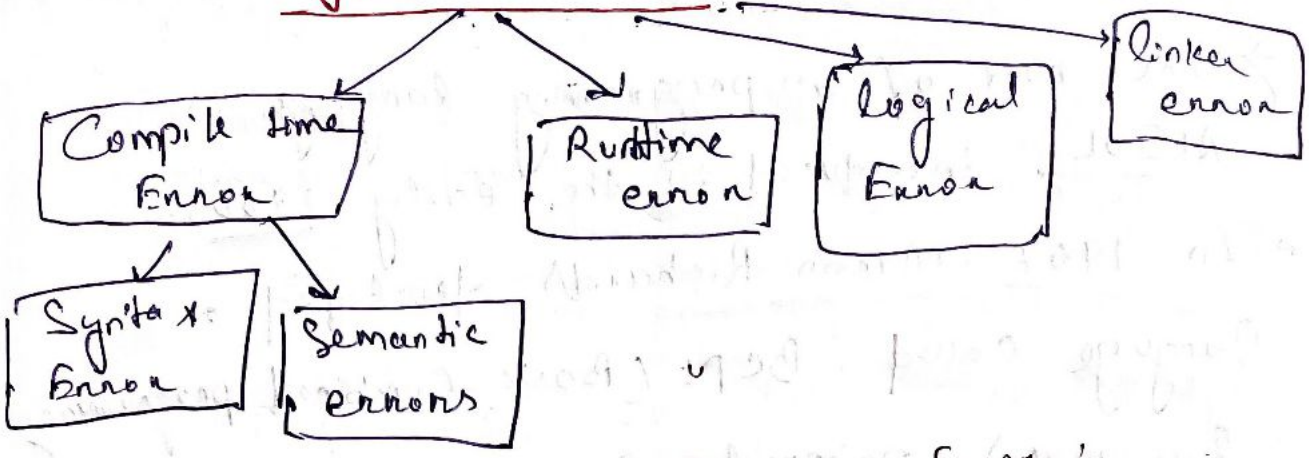
* It is an informal way of programming description that does not require any syntax.

Example (To add two number)

- Declare two variable
- input two variable
- find sum = 1st + 2nd number
- Display output.

Syntax :- Rules to write a program

Types of Errors



① Compile-Time error :-

- Detected before execution during Compilation.

Ex `int a = "hello";`

✗ Data type mismatch

↗ Program does not Compile.

② Syntax Error :-

↗ Occurs when rules of the programming language are violated.

- Ex :- `let a = 10;`
  `Console.log(a);`

✗ - Missing (')

③ Semantic Error :-

↗ The program may compile and run, but the result is incorrect or meaningless.

Ex `int x = 10;`
`x + 5;`

✗ - Result is not used.

↳ Meaningless Statement.

④ Runtime Error :-

↗ A runtime error occurs while the program is executing.

Ex
```
int arr[3] = {1, 2, 3};
printf("%d", arr[5]);
```

✗ - Invalid array index.

⑤ Logical Error :-

↗ Program Executed Successfully but gives wrong Output (Unexpected ~~error~~ Output)

Ex find Sum :-

JS →
```
let a = 10;
let b = 20;
Console.log(a - b);
```
// Wrong Logic
Should be (a + b).

⑥ Linker Error :- Occurs before execution and after compilation.

Ex - If we miss function definition in C lang. then fun will not execute.

'C' language

→ The root of all programming language is ALGOL, introduced in the early 1960s.

→ In 1967, Martin Richards developed a language called BCPL (Basic Combined programming language) primarily for writing System Software.

→ In 1970, Ken Thompson developed "B" language using features of BPCL.

→ "B" was used to create early version of "Unix" operating System at Bell laboratories.

→ 'C' was developed from ALGOL, BCPL & B by Dennis Ritchie at Bell laboratories in 1972.

→ In 1989, ANSI Commitee approved a version of C as a standard language which is known as ANSI C.

Basic Syntax of 'C'

```
#include <stdio.h>  // header file

int main()  // Main function
{
// Variable declare
// body
// Return value
}
```

(Each statement ends with semicolon(;))

Header file :- → In 'C' header file extension is '.h'.

Ex - stdio.h → Standard input & output fun^n.

E.g. Scanf(), printf() etc

- Conio.h → Console input & output fun^n.

E.g. Clrscr(), getch() etc

- String.h → String handling fun^n

- Math.h → Common Mathematical fun^n

→ #include → #-preprocessor directive which include different header file.

→ Print Hello

```
#include<stdio.h>
Void main(){
  printf("Hello");
}
//output- Hello
```

```
#include<stdio.h>
int main(){
  printf("Hello");
  return 0;
}
```

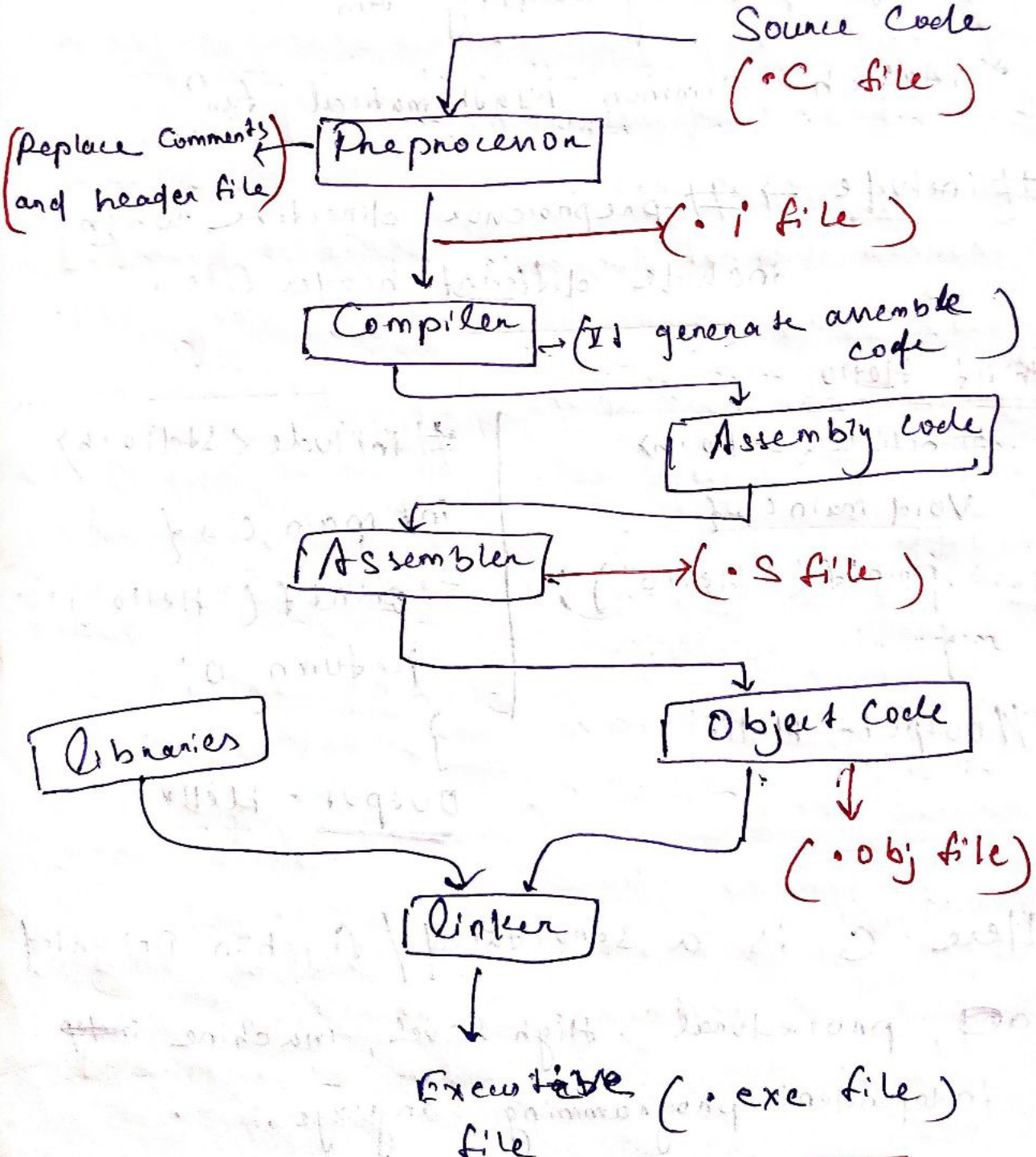
output - Hello

→ Here 'C' is a structured / function Oriented, procedural, High level, machine independent programming language.

→ It follow Top down modular approach (in which we devide the program on the basis of functions).

→ 'C' is highly portable. we need a compiler "gcc" for compile .C file.

Compilation process of 'C'



→ To compile .c file.

gcc .c filename → for Compile

./a.exe → for Output

- Comments

// → Single line Comments

/* .... */ → Multiline Comments

- ~~Abcd~~

Character Set :-

- 'C' Support total 256 Characters.
  - Alphabet (A-Z) (a-z)
  - → ① Digits (0-9)
  - Special symbol (#, $, %. - - -)

Variables

→ Variable is an identifier that is used to hold some value in memory location.

- A variable is a Container (storage area) that is used to hold data.

Naming Convention :-

① → Variable Not a reserved keyword name.

keyword :- Reserved word, value knows to Compiler.

- Total 32 keyword in 'C'.

② → Not longer than 31 Character.

③ follow identifier ruler

① Declaration

① Datatype var-name ;

Initialization

int a = 5; / int a;
a = 5;

① Datatypes

- ① Datatypes are the keywords that describe the type of data that a variable can store, the type of data that a fun can return.

① Datatype

- Primitive (Built-in datatype)
  - Integer - 4 byte (int) (-2^{31} to $2^{31}-1$)
  - float - 4 byte (fractional value) (float)
  - ① Double - 8 byte (double)
  - Character - (char) - 'A'
  - Void
- Non-primitive (Referenced datatype)
  - ① Derived
    - Array
    - Pointer
    - function
  - Userdefined datatype
    - Structure
    - Union
    - Enum
    - Typedef

Identifiers

> Are the names given to Variable, Constants, fun^{n} & User-defiend data types.

- Identifier must be Unique.

Rules for naming identifiers :-

1- 1st letter should be either a letter or underscore.

2- Can't use keyword

3 - Case-sensitive.

4 - comma and backspace are not allowed within an identifier

Tokens :- Smallest element of a program that are meaningful to the Compiler.

- Various types of tokens are

1. Keywords
2. Identifiers
3. Literals – A Constant value can be assigned to the Variable.

   Ex `int x = 10` → Constant/literals
   (int: datatype, x: var-name)
   Integral, floating, char
4. Operations
5. Constant
6. Delimiters

→ Are separators or punctuations.

- Comma(,), semicolon(;), Colon(:), ( ), [ ], ' ', " ", { }, # etc.

Basic Structure of a C program

Step-1 Documentation
(Give the details associated with the programs)

Step-2 Link section
(used to declare all header files)

Step-3 Definition Section
(Define different Constants)

Step-4 Global declaration Section.

Step-5 Main function

Step-6 Sub program Section.

```
/* WAP to find the area of Circle */        → Documentation
#include<stdio.h>                            → Link Section
#define PI 3.14                              → Definition section
void circle();                               → Global declaration section
float r;
Void main() {                                → Main fun
  printf("Enter the radius:");
  Scanf("%f",&r);
  Circle();
}
Void Circle() {                              → Sub program section
   float area = PI * r * r;
  Printf("Area is %f", area);
}
```

Constant:-

- A fixed values that the program can't alter/modify during its execution.
- There fixed values are called as literals.

① Define a Constant:-

→ `# define PI 3.141`

→ `const int a = 3;`

Syntax

```
# define <identifier> <value>
Const <datatype> <variable> = value;
```

Escape Sequences

\n = new line

\? = Question mark

\b = Backspace

\\ = Backslash

\' = Single quotation

\" = Double quotation

\t = insert a tab/space in the text at this point.

Input & Output

Output printf("Hello world");

Cases

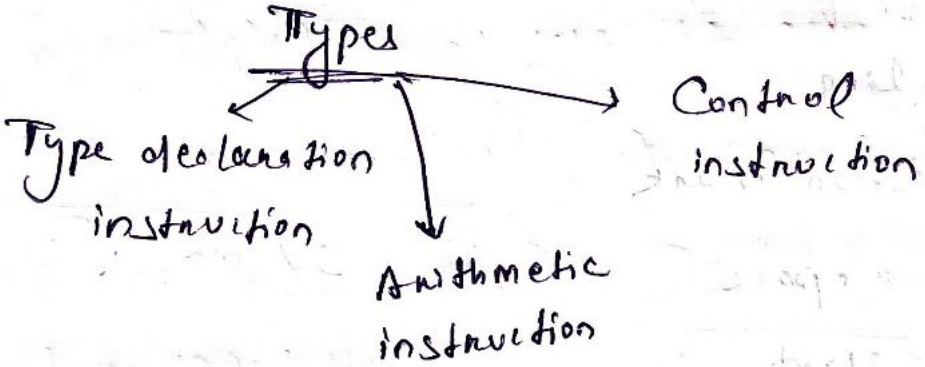
1) integer printf("age is %d", age);

2) Real number printf("age is %f", age);

3) characters printf("star looks like %c", star);

Input Scanf("%d", &age);

Instruction :- These are statements in a program.



① **Type declaration**

Declare variable before using it.

② **Arithmetic instruction** :- (+, -, *, /, %)

Ex: a + b → operand-2.

operand-1 ↙ ↓ operator

Note

** → Single Variable in LHS

a = b+c ✓

b+c = a ✗

Note pow(x,y) for x^y.
by using "#include <math.h>"

Modular operator (%) ② Division (/)

4 % 2 → 0 (Remainder)

4/2 → 2 (Quotient)

③ Control instruction :-

- Used to determine flow of program.
- It is of four type.
- ① Sequence Control instruction
  ② decision " "
  ③ loop " "
  ④ Case " "

Operators

→ Operators are the symbols that are used to perform some mathematical and logical operations.

→ Types of operator.

① Arithmetic operator
② Relational operator
③ Logical operator
④ Bitwise operator
⑤ Assignment operator (Shorthand operator)
⑥ Conditional operator (Ternary operator)
⑦ Size of() operator
⑧ Increment & Decrement operator (Unary operator)

① Arithmetic operators

(+, -, *, /, %)

② Relational operator

== → equal

!= - Not equal

< - less than

> - greater than

<= - less equal

>= - greater equal

③ Logical operator

&& → AND - (Both true)

|| → OR - (Only one true)

! → NOT - (Reverse the result)

④ Bitwise operator (Bit-by-Bit operation)

① Bitwise - AND

A = 1 0 1 0

B = 1 0 0 1

1 0 0 0 = 8

② Bitwise OR

A = 0 1 1 0

B = 1 0 1 0

1 1 1 0 = 14

③ Bitwise Complement (~A)

A = 1 0 1 0

~A = 0 1 0 1

④ Bitwise XOR '^'

A = 0 1 0 1

B = 1 0 1 0 (Same 20 other 21)

1 1 1 1

= 15

e) Switch statement

```
switch (variable) {
    Case Value 1 :
            // statement ;
            break; / ~~continue ; / Jump ;~~
    Case value 2 :
            //statement;
            break; / ~~continue~~;
    Case value 3 :
            // statement ;
            break ;
    default : {
            // statement
        }
```

Ex

```
int main () {
int day ;
printf ("eter day 1-7 ")
scanf ("%d", &day) ;
Switch (day) {
 Case 1 : printf ("Monday \n ");
          break ;
 Case 2 : printf ("Tuesday \n ");
          break ;
    ;
```

⑧ Assignment operator / Shorthand operator

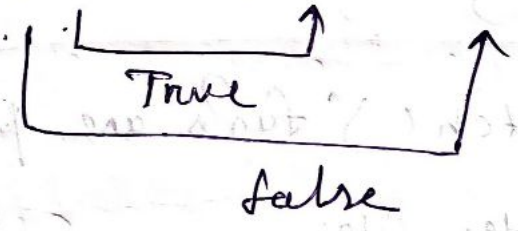
=, +=, -=, *=, /=, %=.

a = a + b	b = b - c
⇒ a += b	b -= c

⑥ Conditional / Ternary operator

(Cond^n) ? statement 1 : statement 2;

Ex: a > b ? a : b;



⑦ Sizeof() operator

- This operator returns the size of a variable.

Syntax

Sizeof (variable name);

Ex: int a;

int size;

size : size of (a);

return : 2 byte

(8) Unary / Increment & Decrement operator

⇒ a++ ⇒ a = a+1 ⇒ a += 1 ⇒ a++

(Increment post)

⇒ ++a ⇒ a
(pre increment)

⇒ a-- ⇒ a = a-1 ⇒ a -= 1 ⇒ a--
(post decrement)

⇒ -a
(pre decrement)

(3) clrscr() & getch() fun's are provided by <conio.h> header file.

⇒ clrscr() fun ⇒ Clear Screen

⇒ getch() fun ⇒ To hold screen.

Operator precedence

1 — !

2 — *, /, %.

3 — +, -

4 - <, <=, >, >=

5 - ==, !=

6 - &&

7 - ||

8 - =

When precedence is same we use associativity rule means role from left to right

Type Conversion

Implicit Conversion (Coercion)	Explicit Conversion (Casting)
→ Done automatically by compiler.	→ Done manually by user.
→ Int to float.	→ float to int.
→ No data loss.	→ Data loss.

→ int a = 2;
float b = a;

Decision Control Structure:

① Decision Making Structure :-

a) Simple if statement

```
if ( condition ){
    // statement will execute if cond is true.
}
```

b) If... Else statement

```
if ( condition ){
    // cond is true
}else{
    // statement will execute
       if cond is false.
}
```

c) Nesting if-else statement

```
if ( cond ) {
        if ( cond ) {
            //———
        } else {
            //———
        }
    }
    else {
        //———
    }
```

d) Else if ladder

```
if ( cond-1 ) {
        //———
}
else if ( cond-2 ) {
        //———
}
else if ( cond-3 ) {
        //———
}
else {
    //———
}
```

```
    }
    Case 7 : printf("Sunday\n");
             break;
    default : printf("\nNot a valid day\n");
  }
  return 0;
}
```

② Looping structure

① for loop :-

```
for( int i=1; i<=8; i++){
    //——
}
```

Syntax

```
for (initialisation ; Condition ; updation ){
    // Do something
}
```

② while loop

```
initialization ;
while ( Condition) {
    // Do something
    updation ;
}
```

③ Do while

```
initialization;
do {
    // Do something
    updation;
}
while (condition);
```

Jumps in loop

- Break
- Continue
- Goto

① Break (Exist from the loop).

- Used in loops & Switch Cases

Syntax `break;`

② Continue (Skip to the next iteration)

- Syntax `continue;`

③ Goto statement