# C Programming

1. Theory of C → Importance
→ properties
Structure of C
→ Escape Sequence.
→ Data type.
→ Modifier
→ C tokens. — Identifier
- Constant
- Keyword
- Operators

## Operators

- Arithmetic
- Assignment
- Relational
- logical
- Increment / Decrement
- Bitwise
- Conditional
- others.
- Terminary operator.

→ Flow chart
→ Algorithm

- Selection Statement
  - Simple if
  - if else
  - neste if
  - if else ladder
- Switch case
- Iterating statement (loops)
  - Unconditional Jump [ Goto ]
  - Conditional Jump [ while / Do while etc)
    - for loop
    - while loop
    - Do while loop

* Nested for loop

- Array, (i) One-D
  (ii) two-D

Searching
Sorting ( Ascending / Descending)
Max / Min value
Inserting value
Deleting value
Updating value
Reverse Array

* Sum of two metrics
* multiply of two metrics
* Transpose of metrics.

(8)

## Strings

- Reading
- Writing
- Combining
- Copying
- Comparing
- Extracting

## Function

Declaration / Definition & Call

- No argument, No return
- No argument But return
- argument But No return
- argument & return.
- Recursive function
- Call by value.
- Call by refrence
- Array as parameter.

## Pointer

## Union

Declaring.
Initializing

## Structure

- Structure Pointer
- Structure Array
- Structure Union
- Structure as argument

## File handling

- Open
- Close
- Read
- write

## Command line argument

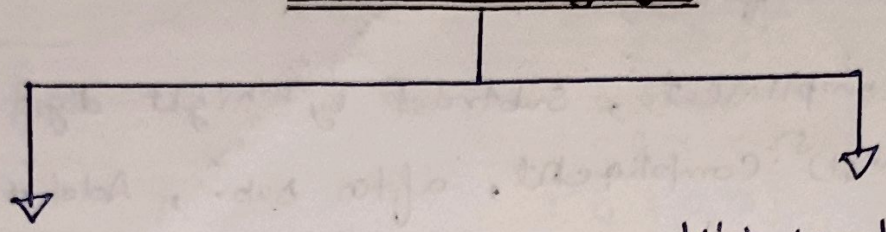
## Storage Classes

Auto
Extern
Static
Resister

## Enum

Defination
&
Program

# Computer language



## Low Level Language (LLL)

-> LLL are machine code or close to it.

-> Computer only execute LLL or binary (0 or 1)

* **Machine language**
  lowest & elementary PL which CP understands.
  0 means absence of ele. impulse
  1 → Peresenc of ele. impulse
  -> fast and efficient use
  -> no requires translator
-> All code have to remember
-> Machine dependent

* **Assembly language**
  in Alphanumeric symbols.
  (mnemonic code)
  combination of max 5 letter.
  (Symbolic PL)
  easy to locate & correct error
  modified easily.
-> machine dependent
-> need hardware knowledge.

## High level language (HLL)

-> in English words & mathematical symbols.
-> need a translater
-> perforns different variety of task.
-> User friendly.
-> easier to learn.
-> easier to maintain.

**Q Why we need Programming language?**

In order to communicate with computer user also needs to have a language that should be understood by the computer.

3/09/2024

Ctrl + Alt + T → Dosbox => Turbo C (open)

1st experiment:- to learn about c Library, Pre-Processor directive, input-output statement

```
#include<stdio.h>
#include<conio.h>
Void main()
{
clrscr();
printf("Hello world")
getch();
}
```

Stdio — Printf()
\ Scanf()

() → Peranthesis

# => Pre processor

# include / # define => Pre-Processor directive.

~~#include~~

Stdio => Standard input output

Conio => console input-output

< > => Angle brackets

→ C programming language was developed in 1972

Developer → Dennis M. Ritchie.

FORTRAN → COBOL → BCPL → CPL → B → C;
(Maths) (Business)

General purpose programming language.

Uses → Unix, Linux, Windows, Mac, Android etc.

→ Data base softwares → Oracle, MySQL

→ Text editor => Notepad, wordpad, Microsoft office etc.

→ Device drivers = Printer, Scanner, Phone, Pen drive etc.

→ Major part of web browsers like google, Chrome, Internet explorer etc.

→ Virus and antivirus

→ Robotics, IoT (Internet of things)

→ Graphics Packages

## ○ Features / Characteristics of C language :-

### Features

(i) General purpose programming language (PL)
(ii) Powerful / Robust PL
(iii) Modular / Procedure oriented P.L (Top to down Process)
(iv) Structured P.L.
(v) Simple and Small programming lang.
(vi) Middle level P.L.
(vii) System P.L.
(viii) Portable machine independent programming lang.
(ix) Extensible, Format free. P.L.
(x) Case sensitive P.L. (xi) Dynamic memory management
(xii) Pointers (xiii) Mother P.L.
(xiv) Strongly / Staticly typed P.L

Case sensitive
means
Upper case and lower case are diff° &
~~they are d~~.
like A ≠ a
they both are diff

* Structure
Documentation / Commenting
For
* Header files inclusion.
Macro Definition (if any)
Global variables declaration
* Main ( ) Function
function 1 ( )
function 2 ( )

⇒ "Const" ⇒ & #define ⇒ Declaration of Constant
#define PI = 3.14 .
Const 3.14 ;

→ C language has 32 keywords.
it make easy to understand.

→ C is known as mother programming language.
C++, Jave, C# are derived by C

In P.L Language

Letters → character set
words → Tokens (Smallest element)
Sentences → Instruction (by : syntax rule)
Paragraphs → Programs

C- Character set

- Alphabets → Upper case ⇒ A to Z
  → Lower case ⇒ a to z
- Decimal digit → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- special symbol → +, -, =, (,), $, {, }, [,], *, ;, :, /, \, <, > etc.
- White Spaces → Blank spaces, Tab space, \n, \t, \r \a etc

C Character set supports ⇒ ASCII
(American Standard code for Info. Interchange)

ASCII Character size = 8 bit

## ASCII Value

| | |
|---|---|
| 0 | Null Character \0 |
| 32 | Blank space |
| 48-57 | for 0 to 9 |
| 65-90 | A to Z |
| 97-120 | a to z |

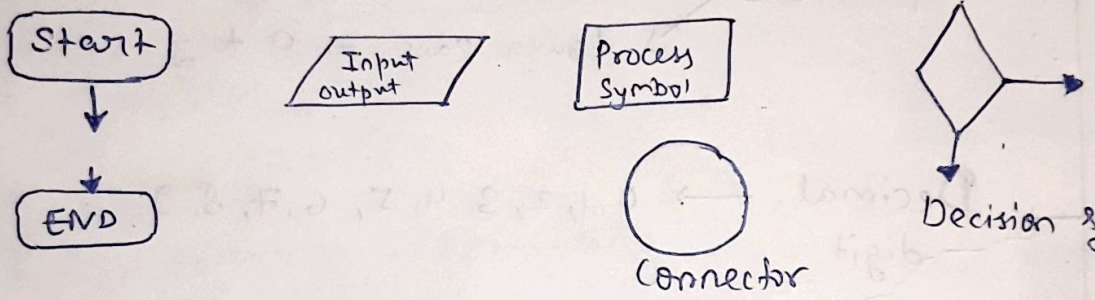
ex.
(0→48, 1→49, 57→9
A→65, B→66, Z→90)

**Compilation** The process of converting source code into machine code (00r 1)

Compiler → also detects error in source code.

library function → There are many built in function available in C prgL. These are stored in diff. Header file.

**Flowchart** The pictorial representation of algorithm.
↳ expressed in image)




**Keywords** a reserved word.
Fixed meaning
Always written in lower case.
(32)
(cannot used as variable)

**Identifies** → names of variable / constant / function
↳ user defined names.
SUB, num1, STDNAME

**Datatypes**
→ To store data program must reserve space which is done by datatype.
predefined instruction
used for allocating memory for data.
Declaration of variable

**Variable** → they may very their value.
↓ (Symbolic)
name of memory location.
Used for store data.
variable are nothing but identifier.
decleration must be done before using it in program.

Date → 3/Sept/2024 [PPS]

Tools: Algorithm → finite set of sequence.
Flow chart → General solution.
Code in lang. → Programming Construct.

→ Focus on input set, process, output for desired.

→ Sequence ⇒ up to down process
Selection ⇒ working for particular steps.
Itration/Repetition ⇒ Repeating the steps again & again.

Datatype :-
According to situation of programming.
like numerical, Alphabates
Variable According to size → capital or small Alphas.

Date 6 Sept 2024

→ Case sensitive language ⇒ Small or capital letters are different from each other.

→ C is Procedural, structural and Top to down approching language.

→ Structure

```
#include <stdio.h>
#include <conio.h>
Void main()
{
clrscr();
printf("Hello world");
getch();
}
```

*main =>
(i) beginning of execution.
(ii) Sucess status return करना।

Void main()
↳ User defined function

clrscr() → library defined function.

→ Pre Processor directive
  # include
  # define

→ Global declaration = that everyone can use
  (like a = 5)

main function declaration
{
definition of main function
 function 1 {};
 function 2 ();
}

Stdio => Standard Input output

Conio => Consule input output

Clrscr → clear screen (erase करना)
printf (" "); => Stdio का member
; => full stop for c programming language (PL)

* Smallest element in c language is Token
getch => conio library का member
  ch => Character => output screen hold करना।
  → Single character input में लेना।

***

| |
|---|
| Alt + F9 - Compile |
| Alt + F5 - User window |
| Ctrl + F9 - Run |

## * printf *

→ printf("Hello world") then it will print => Hello world

→ printf("\n welcome to the world of \n C programming")

it will print → welcome to the world of
C programming

- \n => for next line

### Escape sequence

\n - new line
\t - tab
\b - audible bell
\r - carriage return

\" - double quotes
\' - Single quote
\? - Question mark
\f - new page/ clear page

H.W

① Print

```
R       R
  E   E
    E
  M   M
A       A
```

②

```
* * * * * * * * * *
FName: Reena
LName: Ajmera
Mob. : _____
Class : _____
Section : _____
* * * * * * * * * *
```

③

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

① R R
E E
E
M M
A A

```
#include <stdio.h>
#include <conio.h>
 void main ()
{
clrscr ();
printf (" R \t\t\t R \n \t E \t E \t \n \t\t E \t\t \n
          \t M \t M \t \n A \t\t\t A ");

getch ();
}
```

② 1
1 2
1 2 3
1 2 3 4
1 2 3 4
1 2 3 4 5

```
#include <stdio.h>
#include <conio.h>
 void main ()
{
clrscr ();
printf (" 1 \n 1 \t 2 \t \n 1 \t 2 \t
     3 \t \n 1 \t 2 \t 3 \t 4 \n
     1 \t 2 \t 3 \t 4 \t 5 ");

getch ();
}
```

(3)

```
* * * * * * * * * *
F Name  : Reema
L Name  : Ajmera
Mob.    : _____
Class   : _____
Section : _____
* * * * * * * * * *
```

```
#include <stdio.h>
#include <conio.h>
 void main()
 {
  clrscr();
  printf("\*\t*\t*\t*\t*\t*\t*\t*\t*\t*\n
          \t F\t Name\t\t: \t Reema\n
          \t L\t Name\t\t\t: \t Ajmera\n
          \t Mob.\t\t\t : \n
          \t Class\t\t\t : \n
          \t section\t   : \n
          \*\t*\t*\t*\t*\t*\t*\t*\t*\t*");
  getch();
 }
```

for representing space on paper => \t
1 tab = space of 8 character.

• **Datatypes**

| | |
|---|---|
| **Primary** Data types | ex. Character, integer, floating, double etc. (↑ higher precision) The most basic datatypes that are used for representing simple values |
| User defined Data types | The user defined datatypes are defined by the user himself. |
| Derived Datatypes | The datatypes that are derived from the primitive or built in datatypes are reffered to as Derived data types. |

**Data types in C**

- Primary
  - Integer
  - Character
  - Float
  - double
  - Void
- Derived
  - Function
  - Array
  - Pointer
  - Refrence
- User defined
  - Class
  - Structure
  - Union
  - Enum
  - Typedef

| Datatype | Size (bytes) | Range | Format Specifier |
|---|---|---|---|
| Short int | 2 | -32,768 to 32,767 | %hd |
| Unsigned Short int | 2 | 0 to 65,535 | %hu |
| Unsigned int | 4 | 0 to 4,294,967,295 | %u |
| int | 4 | -2,147,483,648 to 2,147,483,647 | %d |
| Unsigned long int | 4 | 0 to 4,294,967,295 | %lu |
| long int | 4 | -2,147,483,648 to 2,147,483,647 | %ld |
| Unsigned long long int | 8 | 0 to 18,446,744,073,709 551,615 | %llu |
| long long int | 8 | $-(2)^{63}$ to $[(2)^{63}-1]$ | %lld |
| Signed char | 1 | -128 to 127 | %c |
| Unsigned char | 1 | 0 to 255 | %c |
| Float | 4 | 1.2E-38 to 3.4E+38 | %f |
| double | 8 | 1.7E-308 to 1.7E+308 | %lf |
| long double | 16 | 3.4E-4932 to 1.1E+4932 | %Lf |

- **Modifier**

Signed or Unsigned & short or long
(for requirement of space)
(+) or (−)

- **Declearation**

int num1 num2 ; or int num1 ;
num2 ; }

where int = keyword
num = identifier.
identifier can be variable, constant or function.

**Rules**

1. It should be a to z, A to Z and 0-9 and _ (Underscore) only.
2. It should not start with numeric.
3. No spaces in between the name.
if we want to gap then use underscore
4. 32 characters are significant.

* **Increment / Decrement (by 1)**

```
1  int d = 5
2  ++a ;  6 (decrement)
   a++ ;  7 (increment)
   Printf ("%d", a++);  7
   Printf ("%d", ++a);  9
```

diff. between Compiler & interpreter

| Compiler | Interpreter |
|---|---|
| (i) Translate whole source code | Translate line by line. |
| (ii) Executes faster | it takes time / debug easily. |
| -> Feeback after complete the process. | -> Immediate feedback. |

C token => Smallest term (element) of Programming language.

Three types of C tokens
(i) Identifier.
(ii) Literal (Constant)
(iii) Keyword.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  clrscr();
  int num1, num2, Sum3;
  num1 = 5,   num2 = 10
  Sum = num1 + num2;
  printf("Sum of %d and %d is %d", num1, num2, Sum);
 }
getch();
}
```

Here

Identifier = num1, num2, Sum

Operators => +, -, = etc.

Special symbols => %, ', ', ;, :

* Literal constant :- [ 32 keywords ]

- Numeric (5, 10 etc)
- String "Sum of ... is and :" (inside inverted comma)

Keywords (32)

| | | | | | | |
|---|---|---|---|---|---|---|
| goto | break | union | extern | int | short | void |
| sizeof | continue | typedef | Register | Signed | float | if |
| do | enum | volatile | else | unsigned | double | while |
| return | struct | const | static | long | Char | for |
| Switch | case | default | auto | | | |

```
#include <conio.h>
void main()
{
int n1, n2;
printf("enter two no.");
Scanf("%d%d", &n1, &n2);
if (n1 > n2)
  printf("N1 is greater than N2");
else
  printf("N2 is greater than N1");
}
```

Note

" " → String literal

' ' → Character literal

## Operators

1. Arithmetic operator => +, -, /, *, % (modules)
2. Assignment Operator => =, +=, -=, /=, *=, %=
3. Relational => >, <, >=, <=, ==, != → Comparing (true or false)
4. logical operator
5. Increment / Decrement operators
6. Bitwise operator
7. Other (Conditional, size of etc) operators

## * Shortment assignment

+= a = a+b; or a+=b;

-= a = a-b; or a-=b;

## • Bitwise Operators

`&` → And → ⊗
`:` → OR → ⊕
`^` - XOR
`~` - NOT
`>>` - Shift Right
`<<` - Shift left

**Ex** `int a=60, b=54, c;`

(i) `c = a & b;`

$a = (60)_{10} = (111100)_2$

$b = (54)_{10} = (110110)_2$

`Printf("%d", c);` = 52

Process

```
  1 1 1 1 0 0
& 1 1 0 1 1 0
-------------
  1 1 0 1 0 0   = 52
```

(Hint work as AND gate)

(ii) `Printf("%d", (a : b));` = 62

```
  1 1 1 1 0 0
: 1 1 0 1 1 0
-------------
  1 1 1 1 1 0   = (62)10
```

(iii) `Printf("%d", (a ^ b));`

Hint even time 1 then = 0
odd time 1 then = 1
other wise = 0

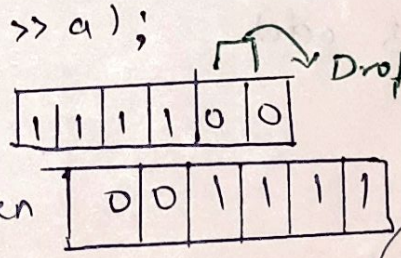
```
  1 1 1 1 0 0
^ 1 1 0 1 1 0
-------------
  0 0 1 0 1 0   = (10)10
```

(iv) `Printf("%d", ~a);`

ex. $(111100)_2 = (000011)$ (once compliment)

(v) `Printf("%d", >> a);`

-> Shift right -

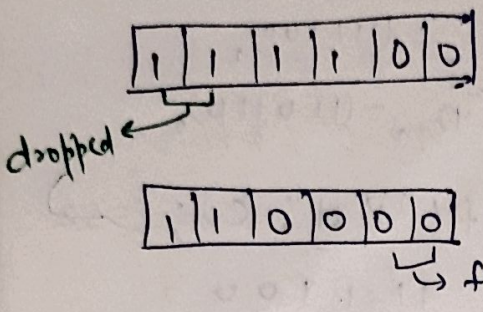



if a >> 2 then

* blank box will be filled up by 0

(vi) << Shift left

if a<<2




## Logic developing

Q ~~Unit~~ Steps for voting condition.

if age greater than equals to 18
then cast vote
else
cant vote.

Q How to check a number is positive, negative or zero.

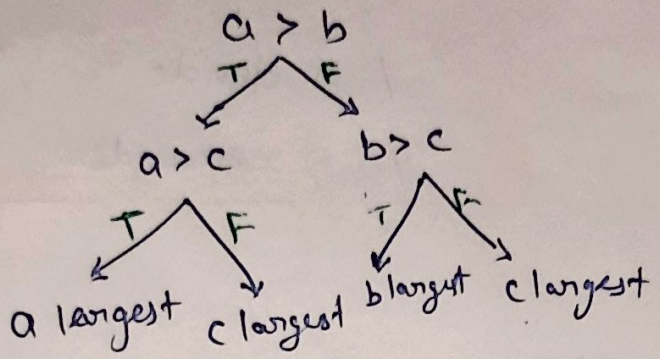
if number is less then 0
then number is negative,
else if number is greater then 0
then number is positive,
else number is zero.

Q How to check even or odd number?

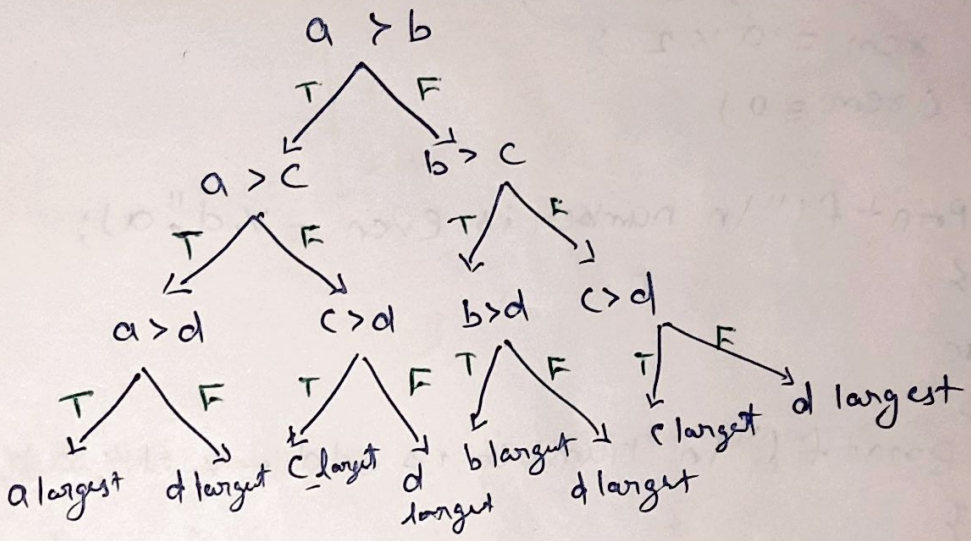
if (n%2) == 0 then
number is even
else
Number is odd.

Q Check greater number.

a, b, c




For a, b, c, d




Q How to check devision.

if percentage is >= 75 then Hons with First class

else if >= 60 && < 75 then First devision

else if >= 50 && < 60 then Second devision

else if >= 40 && < 50 then Pass

else Fail.

20/09/24

## Selection statement

For remainder = int
Otherwise = float

/ => deride
% => remainder

```
# Include <stdio.h>
# include <conio.h>
  void main<>
  $
  clrscr();
  int a, rem;
  printf("Enter a Number");
  Scanf("%d", &a);
      rem = a % 2;
  if (rem = 0)
  {
    Printf("\n number is even - %d", a);
    }
  else
    {
    printf("\n number is odd - %d", a);
    }
  getch();
   }
```

### Types of if

① Simple if

```
if (     )
{

}
```

② if else

```
if (     )
{

}
else
{

}
```

(for a > b)

③ nested if

```
if(      )
  $
  if(      )
    $
    }
    }
  else
    {
    }
    }
```

(for a, b, c or a, b, c, d)

## if else ladder

```
if (          )
{
} else if (          )
{
} else if (          )
{
} else
{
}
```

ladder

if के साथ ; नहीं लगाना है।
Statement में ; लगाना है।

Ex Check the number a is positive, negative or zero.

```
if (a>0)
{
printf("No. is positive");
}
else if (a<0)
{
printf("No. is negative");
}
else
{
printf("No is zero");
```

## Terminary operator (Conditional) :-

(a>b) ? Printf("a is larger then B); : printf("b is larger then a"); ;

① ② ③

↳ Conditional

c = (a>b) ? a : b;

* (a नहीं तो b)
True हो तो c=a
False हो तो c=b

Formate (Condition) ? T : F ;

Ex For odd-even

((a%2)==0) ? Printf("a is even"); : Printf("a is odd"); ;

↳ only for nested if.

For multiple conditions

(Cond) ? (Cond) ? T1 : F1 : (cond) ? T2 : F2 ; ;

T F

* Switch case used when a single varible is used for ~~many~~ different values.

(Ex. in if-else n is used for different values)

```
Switch (varible)
{
  case value 1: Statements;
                Statements;
                break;

  case value 2: Statement;
                break;

  default : Statements;
}
```

break ↳ current block से बाहर हो जाना।

Ex

```
Switch (PocketM)
{
 Case 1; Printf("Shopping");
         break;
 Case 20: Printf("Savings / Shopping");
         break;
 Case 30: printf("mobile");
         break;
 Case 40: printf("Branded clothes");
         break;
 Case 50: printf("Stocks");
         break;
 default : printf("Invalid choice");
 }
```

Ex

```
Switch (pocketM)
{
case 1: printf("Shopping");
        break;
case 10 :
case 11 :
Case 12 :
Case 13 :
case 14 :
Case 15 : printf("Mobile");
          break;
Case 20 : printf("Branded clothes");
          break;
default : printf("Invalid choice");
}
```

EX

Enter choice
1 sum 3 Multiply
2 Divide 4. Subtract
5 Modulus 6. Exit
Enter choice

Alphabate → Sigle quots me

('A')

```
int choice;
if (choice > 0 && < 6)
   Enter two No's
     N1 , N2
Switch (choice)
{
case 1 : printf ("%d", (N1+N2));
         break;
case 2 : printf ("%d", (N1/N2));
         break;
case 3 : printf ("%d", (N1 * N2));
         break;
case 4 : printf ("%d", (N1 - N2));
         break;
case 5 : printf ("%d", (N1 % N2));
         break;
default : printf (" exit ");
}
```

Q Write a program to ~~comp~~ enter the value in cm and convert in km, m and dc.

Soln

```
#include <stdio.h>
#include <conio.h>
  void main()
  {
  float n;
  printf("Enter the value of n in cm");
  scanf("%f=n", &n);
   Switch(n)
   {
   case km: printf("%f", n*100000);
            break;
   case m : printf("%f", n*100);
            break;
   case dm : printf("%f", n*10);
            break;
   default : printf(" exit ");
    }
  getch();
  }
```

* Sum of two numbers

- Sequence
- Selection
- Iteration.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a, b, c;
clrscr();
printf("Enter the value of a, b:");
scanf("%d %d", &a, &b);
c = a+b;
getch();
scanf.
printf("\n Sum = %d", c);
getch();
}
```

- **Iteration / Looping / Repetition**

Iteration

Unconditional Jump

Keyword goto _ label

(Underscore) ↳ place.

[valide Identifiers]

(It is not suggestive.)

For ex.

```
_top:
    printf("Hello world");
goto _top;
```

(Non terminating loop)

(Creates infinite loop)

Conditional Jump.

```
goto _ label
int i = 1
_top:
    i = i+1;
    printf("Hello world");
    if(i <= 10)
    { goto _top; }
    printf("Out of goto
            block");
```

⇒ loop for 10 times and then Jump on next.

goto Somewhere; → (Semicolon terminates the statement)

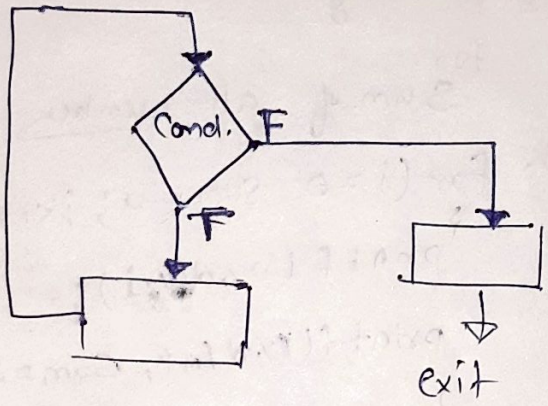
(No execution)

Somewhere:

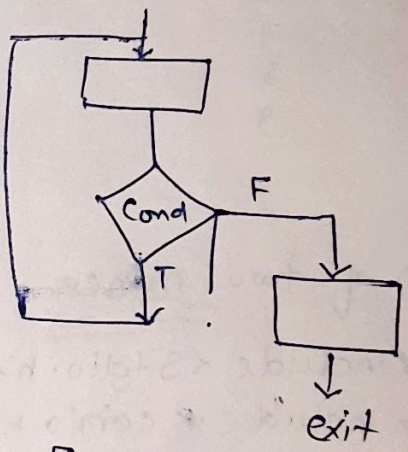
Conditional jump → Entry control
(i) For
(ii) while

Exit control
(i) do-while → (Atleast) One time execution
(ii) ~~while~~

## Flow chart of loops




(a) Entry control




Exit control

(i) For Loop

for (Initialization ; Condition ; increment/Decrement)
(only 1 time) — part of loop.

{

(Execution of code)

}

For example

```
int i;
for (i=0; i<10; i++)
{
 printf("%d", i);
}
```

Output

0
1
2
3
4
5
6
7
8
9

for i++ = 2
⇒ i = i + 2

Output

Even numbers

0
2
4
6
8

Sum of two numers ⇒

```
#include <stdio.h>
# include <conio.h>
void main()
{
 clrscr();
 int i, a, b, c;
 for (i=0; i<10; i++)
 printf("Enter the value of a,b");
 scanf("%d%d", &a);
 printf("
 scanf("%d,%d", &a, &b);
  c= a+b;
 printf("\n Sum = %d", c);
 getch();
}
```

for
Sum of all numbers

```
* for(i=0, Sum=0; i<10; i++)
 {
 printf("%d", i);
 printf("%d\n", Sum = Sum+i);
 }
```

output

0_0
1_1
2_3
3_6
4_10
5_15
6_21
7_28
8_36
9_45
Sum = 45

(Nested For loop)

Ex ==

```
int i, j;
for(i=0; i<5; i++)
{
for(j=0; j<=4; j++)
{
printf("%d", i);
}
printf("\n");
}
```

Output

```
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5
```

if we will put j in printf instead of i then output will be

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 2 | 3 | 4 |

Ex ==

```
for (i=0, i<5, i++)
{
for (j=0, j<=i, j++)
{
printf("%d", i);
}
printf("\n");
}
```

Output

```
0
1 1
2 2 2
3 3 3 3
4 4 4 4
```

if we will put j in printf instead of i then output will be

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
```

Ex
```
for(i=0, p=0 ; i<5 ; i++)
  {
  for (j=0, j<=i ; j++)
  {
   printf ("%d", p);
    if (p==0)
       p=1
    else
      {
       p=0
      }
  }
```

output

```
0
1 0
0 1 0
1 0 1 0
0 1 0 1
```

(ii) While loop

Syntax

```
while ( Condition)
    {
     block of code
    }
```

Ex

```
int i=1;
while (i<=n)
    {
     printf("%d\n", i);
      i++;
    }
```

let n = 5

⇒ output

```
1
2
3
4
5
```

## (iii) Do while loop

Syntax:

```
do
{
//Block of code
}
while (condition);
```

For at least one time execution.

Ex:

```
int i = 0;
do
{
 i = i+1;
 printf ("%d", i);
}
while (i<10);
```

output

1
2
3
4
5
6
7
8
9
10

# Arrays

Collection of similar data types stored at contiguous memory location.

(एक साथ) (Collective)

→ Index number should be always integer.

→ indentifies with index number.

For ex

without array

```
int marks1 = 40;
int marks2 = 50;
int marks3 = 60;
```

with array

```
int marks[ ] = {40, 50, 60};
```

Syntax (Initialization)

```
int marks [3];
char name [10];
float price [2];
```

(3, 10, 2 => Size)

→ Array starts with 0 index.

for input & Output

```
Scanf ("%d", &marks[0]);   => Store 0
printf("%d", marks[0]);    => Show 0
```

Example

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| marks | 34 | 32 | 50 | 5 | 12 | 14 | 19 | 26 | 24 | 30 |

```
int marks[10], i;
```

=> 20 bites will be stored.

Result

```
Marks[0] = 34
marks [5] = 14
```

For single Dimension array.

int marks [4] = {n1, n2, n3, n4}

Ex int marks [4] = {4, 7, 9, 8};

```
for (int i=0; i<4; i++)
  {
  printf (" The value of %d element
    of array is %d \n ", i, marks[i]);
  }
```

output

The value of 0 element of array is 4
The value of 1 element of array is 7
The value of 2 element of array is 9
The value of 3 element of array is 8

- Input of Array
- output
- Searching
- Sorting
- Max value
- Second max value
- Smallest value
- Second smallest value
- Insert
- Delete — location, Value
- Update — location, Value
- Sum of Array

Operations

For 2D array (2x4)

int marks [2] [4] = {{1, 2, 3, 4}, {3, 2, 3, 3}}

```
for (int i = 0; i < 2; i++)
  {
  for (int j = 0; j < 4; j++)
    {
    printf (" The value of %d, %d element of the array
      is %d \n", i, j, marks[i][j])
```

Output

The value of 0,0 element of array is 1
The value of 0,1 element of array is 2
(and so on)

## Searching

// a[i] input देने के बाद

```
int x, i, flag, loc, a;
printf("Enter x to search");
scanf("%d", &x);
flag=0;
for(i=0; i<a; i++)
  {
  if(x==a[i])
     {
     flag = 1; // 1 for true
     loc=i;
     break;
     }
  }
if(flag==1)
   {
   printf("%d found at %d", x, loc);
   }
else
     {
     printf("Element not found");
     }
```

## Sorting [Ascending and descending order]

```
int i, temp
for(i=0; i<10; i++)
   {
   for(j=i+1; j<10; j++)
        {
        if(a[i] > a[j])
          {
          temp = a[i];
          a[i] = a[j];
```

- Maximum value

```
#include <stdio.h>
#include <conio.h>
void main()
{
int arr[ ] = {24, 12, 6, 5, 8, 14, 26, 29, 3};
int n, max;
max = arr[0];
n = sizeof(arr[0]);    // when we don't know about the
                          size of array
for(int i=0; i<n; i++)
    {
     if (max < arr[i])
      {
       max = arr[i];
      }
    }
printf("Array Elements: ");
  for(int i=0; i<n; i++)
      {
       printf("%d", arr[i]);
      }
  printf("\n");
  printf("The maximum value of the array is : %d", max);
  getch();
}
```

// similar method for minimum value.

- For Second largest :-

```
#include<stdio.h>
#include<conio.h>
  void main()
  {
  int arr[10];
  int i, greatest, second;
  printf(" Enter 10 values:");
   for(i=0; i<10; i++)                    //input array
      {
       scanf("\n%d", &arr[i]);
       }
   greatest = arr[0]
     for(i=1; i<10; i++)
        {
        if(greatest < arr[i])    // greatest value
           {greatest = arr[i]
           }
        }
    second = arr[i];
      for(i=0; i<10; i++)
          {
           if(arr[i] < greatest && second > arr[i-1])
             {
              second = arr[i];          // second greatest
                                           value.
             }
          }
      printf(" The greatest number is: %d ", greatest);
      printf(" The second greatest number is: %d;
                                          second);
   getch();
  }
```

(// Similar for second minimum)

- Inserting a value in array

```
#include <stdio.h>
#include <conio.h>
void main()
{
int arr[100] = {0};
int i, x, pos, n=10;
for (i=0; i<10; i++)
    {
     arr[i] = i+1;
    }
for (i=0; i<n; i++)
  { printf("%d", arr[i]);
    }
printf("\n");
}
 x = 50;      // element to be inserted
 pos = 5;     // position at which insetent
 n++;         // increase size
for (i=n-1; i>=pos; i--).      // print
    {
    arr[i] = arr[i-1];
    }
arr[pos-1] = x;      // insent value
 for (i=0; i<n; i++)
    {
      printf("%d", arr[i]); // print updated array
    printf("\n");
    }
 getch();
}
```

- Deleting a value in array

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int arr[100] = {0};
 int pos, i, num;
 printf("\n Enter the number of elements in an
                                array: \n");
 scanf("%d", &num);
 printf("\n Enter %d elements in array: \n", num);
for (i=0; i<num ; i++);
    {
     printf("arr[%d] = ", i);
     scanf("%d", &arr[i]);
    }
printf("Define the position of the array element where
          you want to delete : \n");
 scanf("%d", &pos);
if (pos >= num+1)
    {
     printf("\n Deletation is not possible:");
    }
else
    {
     for (i=pos-1; i<num-1 ; i++);
        {
         arr[i] = arr[i+1];
        }
     printf("\n the resultant array is: \n");
```

```
for(i=0 ; i<num-1 ; i++)
    {
    printf("arr[%d] = ", i);
   printf("%d\n", arr[i]);
   }
  }
getch();
 }
```

* Updating a value in array:-

```
#included <stdio.h>
#included <conio.h>
  void main()
   {
   int i, t, a[0], n, s
   printf("\n Enter the size: ");
   scanf("%d", &n);
   printf("\n enter the values: ");
   for (i=0 ; i<n ; i++),
      {
       scanf("%d ", &a[i]);
      }
    printf("\n Given values are: ");
     for(i=0 ; i<n ; i++).
       {
        printf("a[%d] = %d, i, a[i]);
       }
     printf("\n Enter the position to be updated: ");
      scanf("%d", &t);
      printf("\n Enter the value to be updated: ");
       scanf("%d", &s);
     for(i=0 ; i<n ; i++),
       {
       if(i==t)
        {
         a[i] = s;
        }
       }
     printf("\n Updated value is: ");
       for(i=0 ; i<n ; i++)
         { printf("\n a[%d] = %d", i, a[i]);
          } getch();
```

## Reverse of array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n, i, a[10], temp;
printf("\n Enter the size:");
scanf("%d", &n);
printf("\n Enter the values:");
for(i=0; i<n; i++)
  {
  scanf("%d", &a[i]);          // Input array.
  }
printf("\n Given values are:");
for(i=0; i<n; i++)
  {
  printf("a[%d] = %d", i, a[i]);   // output array.
  }
for(i=0; i<n/2; i++)
  {
  temp = a[i];
  a[i] = a[n-i-1];          } -> Swap value
  a[n-i-1] = temp;
  }
printf("\n Reverse of array:");
  for(i=0; i<n; i++)
    {
    printf("a[%d] = %d", i, a[i]);   // print reverse array
    }
```

## Addition of 2-D Array :-

```
#include<stdio.h>
#include<conio.h>
  void main()
   {
    int m, n, j, i;
    printf("Enter the number of rows & colums of
            the matrics:");
    Scanf("%d%d", &m, &n);
    int a[m][n], b[m][n], c[m],[n];
    printf "Enter the elements of martix A:\n");
      for (i=0; i<m; i++)
        {
          for(j=0; j<n; j++)
              {
              scanf("%d", &b[i][j]);
              }
        }
     printf("Enter the element of matrix B:\n");
       for (i=0; i<m; i++)
          {
           for(j=0; j<n; j++)
             {
             scanf("%d", &a[i][j]);
             }
          }
     //add the matrics
      for (i=0; i<m; i++)
          {
           for(j=0; j<n; j++)
              {
               c[i][j] = a[i][j] + b[i][j];
              }
          }
```

```
printf("The sum of two matrics is.\n");
for(i=0; i<m; i++)
    {
     for(j=0; j<n; j++)
       {
        printf("%d", c[i][j]);
       }
     printf("\n");
    }
  getch();
}
```

(Similar for substract)

## Multiplication of two array :-

Rule

(m,n) x (p,q)

(n=p Should be equal)

```
#include<stdio.h>
#include<conio.h>
void main()
{
int m, n, p, q, i, j, k;
int a[10][10], b[10][10], c[10][10];
printf("Enter the number of rows & columns of
        first matrix: ");
scanf("%d%d", &m, &n);
printf("Enter the number of rows & columns of second
        matrix: ");
scanf("%d%d", &p, &q);
if(n!=p)
  {
   printf("matrix multiplication not possible.\n");
  }
else
  {
   printf("Enter the elements of first matrix:\n");
   for(i=0; i<m; i++)
     {
      for(j=0; j<n; j++)
        {
```

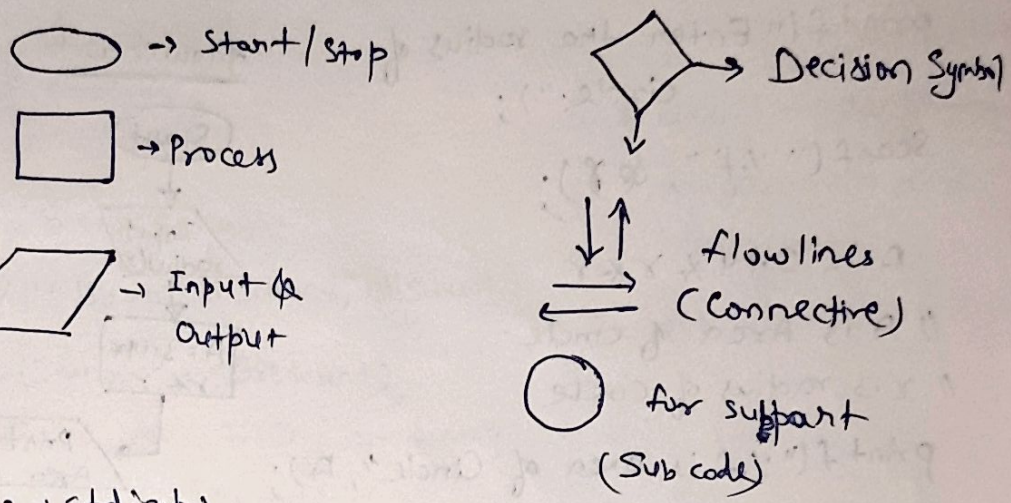
```
        scanf("%d", &a[i][j]);
    }
}

printf("Enter the element of second matrix:\n");
    for(i=0; i<p; i++)
        {
        for(j=0; j<q; j++)
            {
            scanf("%d", &b[i][j]);
            }
        }
            // multiplication
for(i=0; i<m; i++)
    {
    for(j=0; j<q; j++)
    {
      c[i][j] = 0;
     for(k=0; k<n; k++)
        {
         c[i][j] += a[i][k] * b[k][j];
         }
    }
  }

// Print the result.
printf("Resultant matrix:\n");
    for(i=0; i<m; i++)
        {
          for(j=0; j<q; j++)
          {
          printf("%d", c[i][j]);
          }
        printf("\n");
        }
} return ----> else
getch();
}
```

# Algorithms & flowchart

1. **Algorithm** A finite set of sequence or step-by step solution of problem.

2. **Flowchart** general solution in pictorial form.

**Symbols**



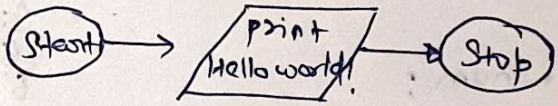

**Ex**

```
#include<stdio.h>
#include<conio.h>
  void main()
   {
   printf("Hello world!");
   }
 getch();
 }
```

→ **Algorithm**

1. Start
2. print("Hellow world!"
3. Stop.

**Flowchart**



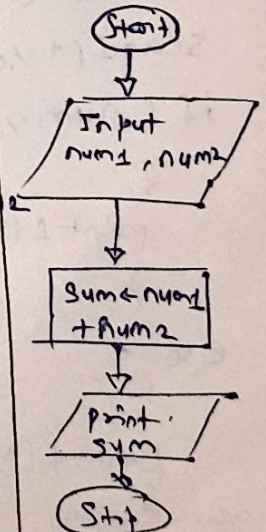

**Ex** Sum of two num.

```
#include<stdio.h>
#include<conio.h>
  void main()
  {
  int num1, num2, sum;
  clrscr();
  printf("Enter the numbers: ");
  scanf("%d %d", &num1, &num2);
  sum = num1 + num2;
  getch();
  }
```

**Algorithm**

1. Start
2. input num1, num2 as integer.
3. Sum ← num1 + num2
4. print Sum
5. Stop.

**flowchart**



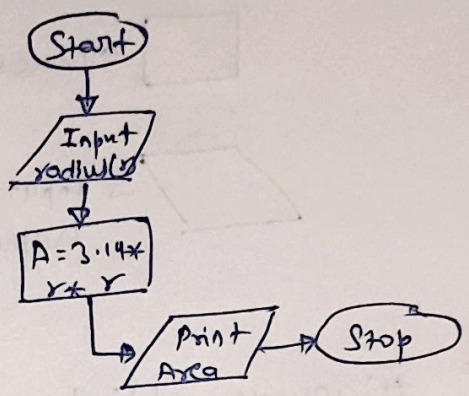
## Ex Area of Circle

```
#include<stdio.h>
#include<conio.h>
 void main()
 {
 float r, A;
 clrscr();
 printf(" Enter the radius of
            circle:");
 scanf("%f", &r);
  A = 3.14 * r * r
 // A is Area of circle.
// r is radius of circle.
 printf("%f is area of Circle", A);
 getch();
}
```

### Algorithm

1. Start
2. Input radius (r)
3. Area = 3.14 * r * r
4. Print Area
5. Stop.

### flowchart



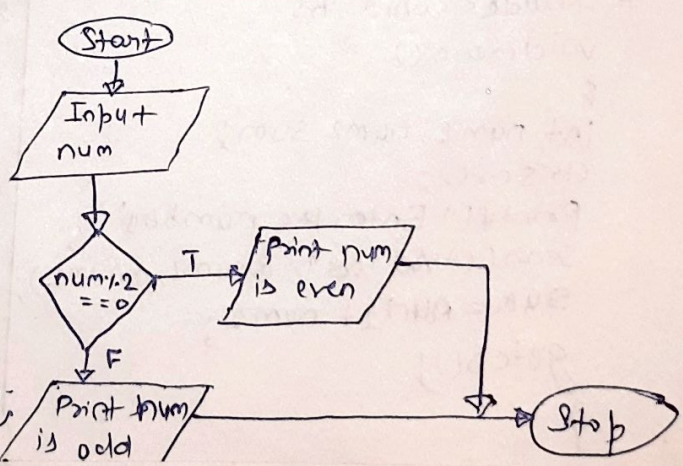

## Ex For odd & even

```
#include<stdio.h>
#include<conio.h>
 void main().
 {
 int num;
 clrscr();
 printf(" Enter the number: ");
 scanf("%d", &num);
 if (num%2 == 0)
   {
   printf(" number is even);
   }
else
   {
   printf(" number is odd");
   }
getch();
}
```

### Algorithm

1. Start
2. input num
3. if num%2 == 0 then go to step 5
4. print num is odd
5. go to step 7
6. print num is even
7. Stop

### flowchart



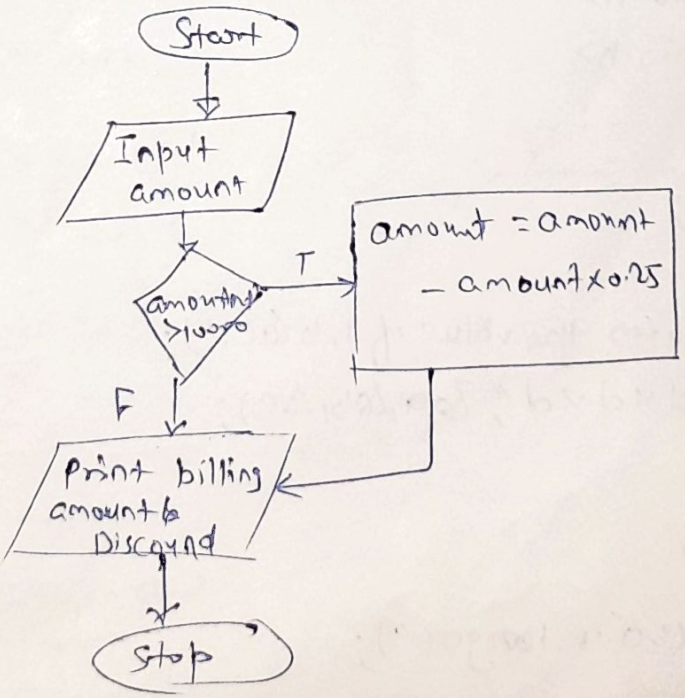
Ex For Discount or billing Amount

```
#include<stdio.h>
#include<conio.h>
void main()
{
float Samount, Discount
printf("Enter the Shoping amount:");
scanf("%f", &Samount);
if( Samount >10000)
 {
 Discount = 0.25*Samount;
 printf("%f"is your discound", Discount);
Samount =(Samount - Discount);
printf("%f is your billing amount";
                    Samant);
 }
else
   {
   printf("%f is your billing amount
                without discound", Samount);
   }
getch();
}
```

## Algorithm

1. Start
2. Input Samount
3. If amount < 10000 then go to step 6
4. SAmount = SAmount - Samount*0.25
5. Discount = 0.25 x Samount
6. print Samount and discount
7. Stop.

## Flowchart



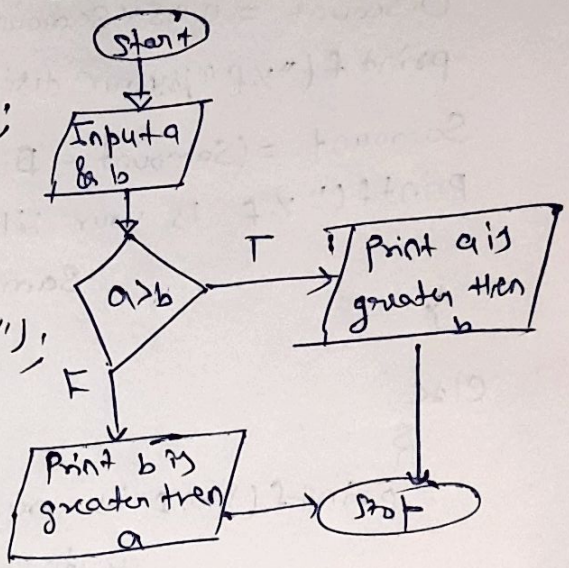
Ex for checking greater b/w a & b

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b;
clrscr();
printf("Enter the value of a & b:");
scanf("%d %d", &a, &b);
if (a>b)
 {
 printf("a is greater then b");
 }
else
 {
 printf("b is greater then a");
 }
getch();
}
```

Algorithm

1. Start
2. input a, & b
3. if a > b then go to step 5
4. print b is greater then a
5. print a is greater then b
6. Stop.

Flowchart




Ex for checking greater b/w a, b & c.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b, c;
clrscr();
printf("Enter the value of a, b & c:");
scanf("%d %d %d", &a, &b, &c);
if (a>b)
 {
 if (a>c)
  {
  printf("a is longer");
  }
```

```
else
    {
    printf(" c is larger");
    }
else if ( b>c)
    {
    printf ("b is larger ");
    }
  else
    {
    printf(" c is larger ");
    }
  getch();
}
```

## Algorithm

(i) Start

(ii) input a, b, c;

(iii) if a > b then goto step (vii)

(iv) if b > c then go to step (v)

(v) print c is larger and go to step (x)

(vi) print b is larger and go to step (x)

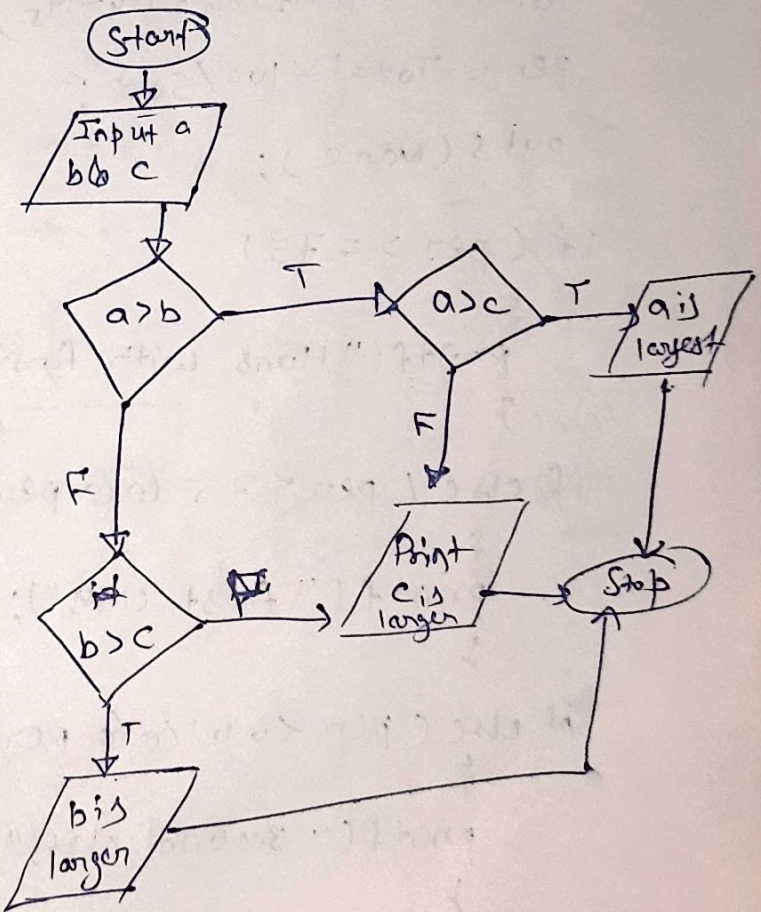
(vii) if a > c then go to step (ix)

(viii) print c is larger and go to step (x)

(ix) print b is larger and go to step (x)

(x) Stop

## Flowchart



Ex

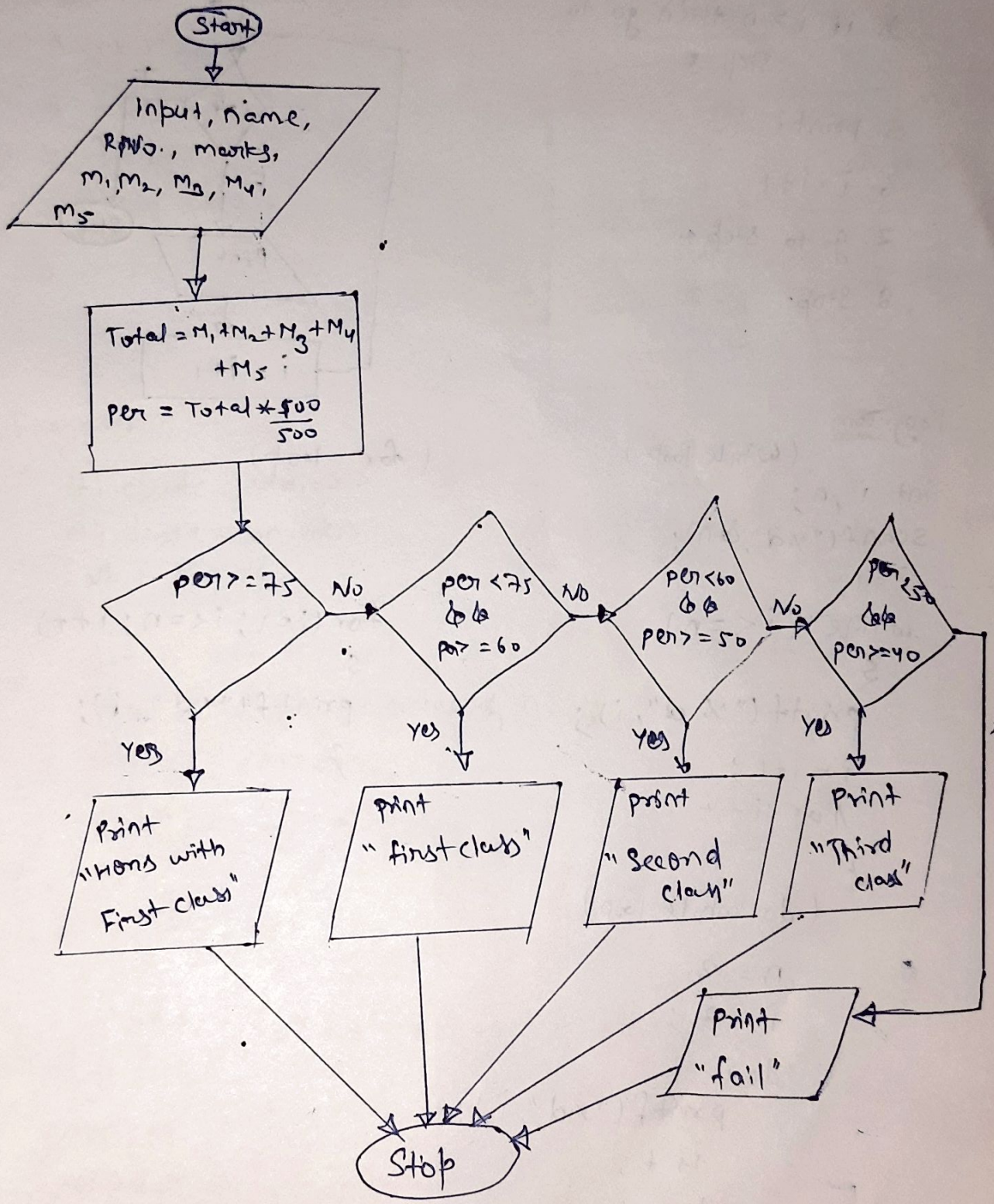
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
  void main()
   {
   char name[50];
   int RNo;
   float M1, M2, M3, M4, M5, total
                per;
   scanf("%d", &RNo);
   gets(Name);
   scanf("%f %f %f %f %f",
          M1, M2, M3, M4, M5);
   Total = M1+M2+M3+M4+M5;
   per = Total*100/500;
   puts(Name);
  if (per >= 75)
     {
     printf("Hons with first class");
     }
  if else (per < 75 && per >= 60)
     {
     printf(" first class");
     }
  if else (per < 60 && per >= 50)
     {
     printf(" second class");
     }
  if else (per < 50 && per >= 40)
     {
     printf(" Third class");
     }
```

## Algorithm

1. Start.
2. Input name as string.
3. input R.No as integer.
4. Input marks $M_1$ $M_2$ $M_3$ $M_4$, $M_5$ as float
5. Total = $M_1 + M_2 + M_3 + M_4 + M_5$
6. per = Total * 100/500
7. if per >= 75 then go to step
8. if per < 75 && per >= 60 then go to step
9. if per < 60 && per >= 50 then go to step
10. if per < 50 && per >= 40 then go to step
11. print "fail" and go to step 16
12. print "Hons with First class" & go to step 16
13. print "first class" & go to step 16
14. print "Second class" & go to step 16
15. print "Third class" go to step 16
16. print "Stop"

```
else if
    {
    printf(" Fail");
    }
```

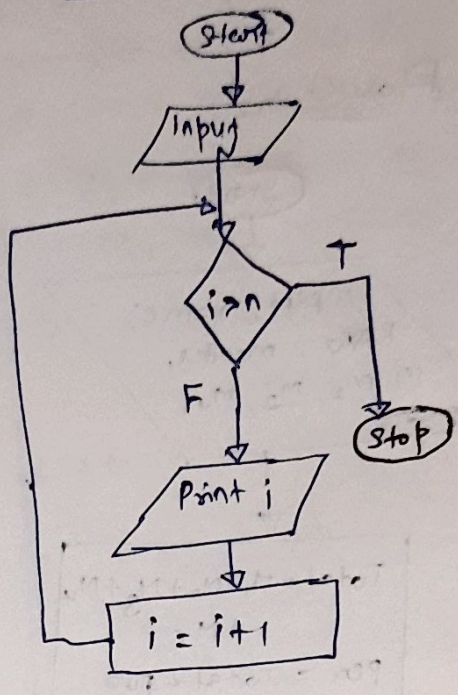
## Flowchart



# For loops (Iterative statements)

Ex 1. Start
2. Let i = 1
3. input n
4. if i > n then go to step 8
5. print i
6. i = i+1
7. go to step 4
8. Stop.

Flowchart




Program

(while loop)

```
int i, n;
scanf("%d", &n);
i = 1;
while (i<=n)
{
printf("%d", i);
i = i+1;
// or i++
}
```

(for loop)

```
for(i=1; i<=n; i++)
{
printf("%d", i);
}
```

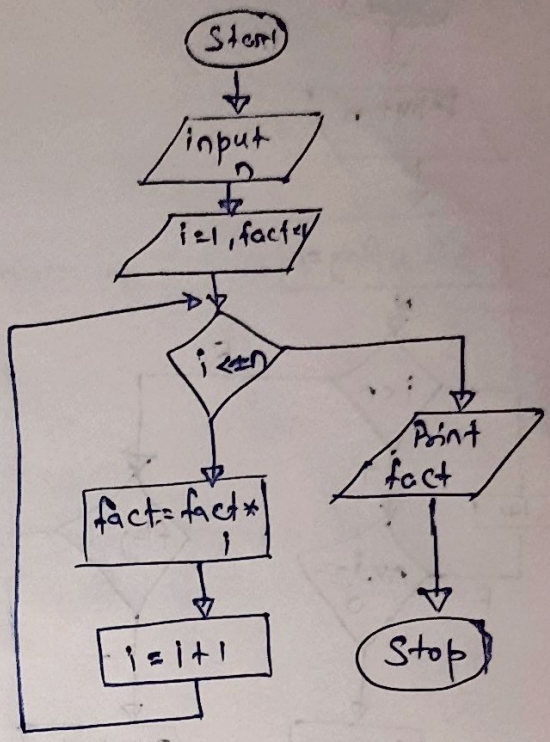
(do while loop)

```
n = 20
i = 2;
do{
printf("%d", i);
i++;
}
while(i<=n);
```

For factorial

1. Start
2. Input n
3. i=1, fact=1
4. if i>n go to step 9.
5. print fact
6. fact = fact * i
7. i = i+1;
8. go to step 4
9. Stop.

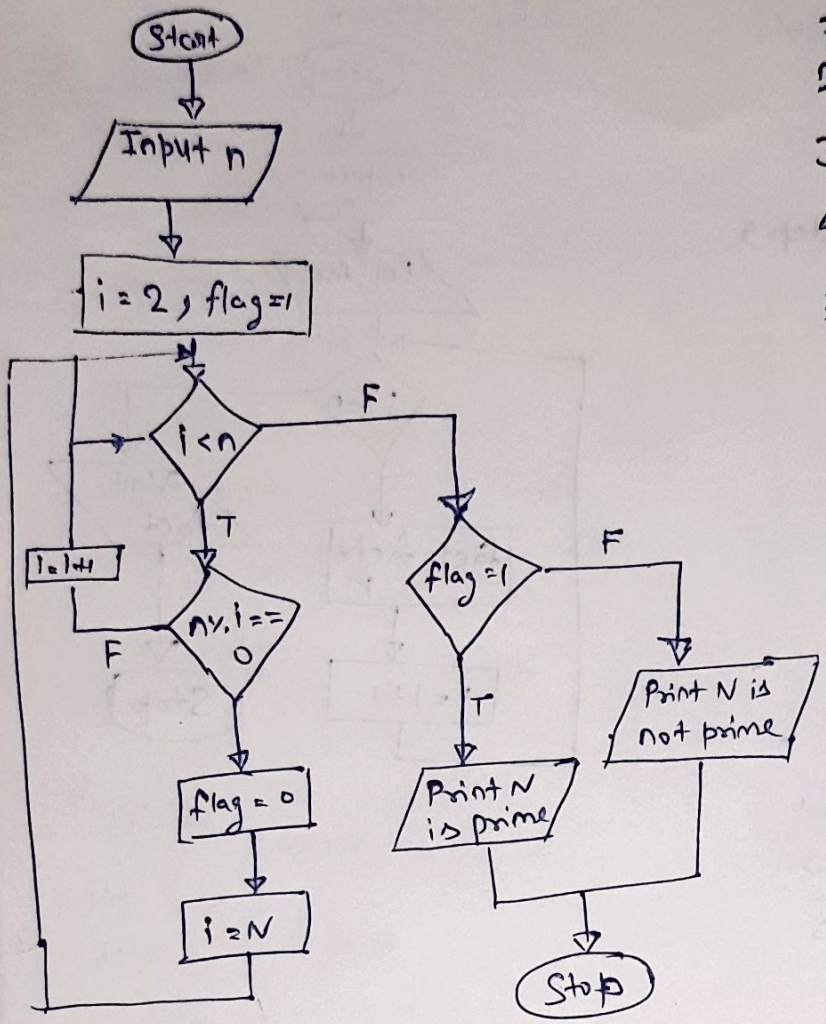
flowchart




Program

```
#include<stdio.h>
#include<conio.h>
 void main()
  {
  int n, fact=1, i=1;
  printf("Enter the value of n:");
  scanf("%d", &n);
  for(i=1; i<=n; i++)
    {
     fact = fact * i
     }
   printf("%d is factorial of %d", fact, n);
  }
getch();
void main.
}
```

# For Prime number




## Algo

1. Start.
2. input n
3. i = 2, flag = 1
4. if i < n go to step 9
5. if n % i == 0 go to step. 8
6. flag = 0
7. i = N go to step 4
8. i = i + 1 go to step 4
9. ~~flag = 1~~
9. if flag = 1 go to step 11
10. print N is not prime. go to step 12
11. print N is prime.
12. Stop.

## Program

```
}
int i = 2, flag = 1
    for (i = 2; i < n; i++)
        {
         if (n % i == 0)
           {
            flag = 0
            i = N
           }
         else
           {
            i++
           }
    }
if (flag == 1)
  {
  printf(N is prime
  printf("%d is prime", n);
  }
else
  {
  printf("%d is not prime", n);
  }
getch();
}
```

# Strings

"Character array"

→ Any group of characters defined between double quotation marks is a string constant.

→ Sequence of characters treated as a single data item.

Common operations of strings
* Reading & writing strings
* Combining strings together.
* Copying one strings to another.
* Comparing strings for equality
* Extracting a portion of a string.

**Declaration**

char string_name [size]

Eg,
char city [10];
char name[5]

**Note**

Size = max(n) + 1

It requires a null char ('\0') in end of the string. (Null char)

for ex "NEW YORK" = char city [9]

↳ {'N','E','W',' ','Y','O','R','K','\0'};

* **Reading string**

by Scanf

char name [10]
~~scanf("%s",address~~
scanf("%s", name);
(no & required)

For tab or gap

char name1[5] name2[5]
scanf("%s %s", name1, name2);
if we use "%ns"
then it will read
'n' char. only.
(n = number)

For read line

char line [80]
scanf("%[^\n]", line);
printf("%s", line);

Using getchar & gets function

① char ch;
ch = getchar( );

② char line [80];
gets (line);
printf("%s", line);

* Writing string

Using printf

printf("%s", name);

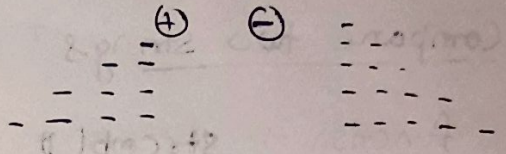
%.ns => prints first n char of string

printf("%*.*s\n", w, d, string);

d => first chara to be print.

w => width

%*.1s



Using putchar & puts

Char ch = 'A';
putchar(ch);

& char line[80]
gets(line);
puts(line);

for Name

char name[6] = "PARIS"

for (i=0; i<5; i++)
{
putchar name[i];
}

* Arithmetic Operations

to write a char in its integer representation -

x = 'a';
printf("%d\n", x) (output => 97)

it can be used as

x = 'z' - 1; ('z' = 122)

(output = 121)

Relational

ch >= 'A' && ch <= 'Z'

Checking whether it is upper case or not.

x = ASCII value of '7' - ASCII value of '0'

x = 55 - 48 = 7

* **putting String together**

Concatenation of string.

Ex String1 = 'Ram'
String2 = 'Sharma'

function => strcat( )

strcat(String1, String2);

strcat(String1, string2);

output Ram Sharma.

* **Compare two strings**

function => strcmp( )

strcmp("their", "there");

[0 or less then 0]

it will return -9

diff between ASCII value

* **Copies one string over another :-**

function => strcpy( )

strcpy(String1, string2);

↳ String2 = String1;

* **Find length of String :-**

n = strlen(String);

DELHI

↳ n = 5

Ex
```
char star[52];
gets(str);        //input
printf("%s", str);
puts(str);        //Print
```

It not include null Character(\0)

**typedef** give alternative name to existing variable.

typedef <previous_name> <alternative_name>;

Ex typedef unsigned long ul;

# * Function *

→ Functions are used to devide a large C program into smaller pieces.

→ A function can be called multiple times to provide reusability & modularity to the C program.

→ Also known as procedure & subroutine or subprograms.

**Syntax**

```
return_type function_name (data_type parameter1, data_type parameter2...)
{
// Code to be executed
}
```

(return_type: prototype & signature; parameter: Argument)

Top to Down modular programming.

ex `int Sum (int a, int b);`

(int: return type; Sum: function name; a, b: parameter Name; int: Parameter type)

## * Advantage

(i) We can avoid rewriting same logic through function.

(ii) We can devide work among programmers using functions. Save time & space.

(iii) We can easily debug a program using function

## * Declaration, Definition & Call

- A function is declared to tell a compiler about its existence.
- A function is defined to get some task done.
- A function is called in order to be used.

call by ref. or call by value

## * Types of Function

C Function

**Library Function**
- → Provided by C
- → built-in function
- → Pre-defined function (in Header files)
  printf / scanf

**User Defined function**
- → Created by user. (to reduce complexity)
  main, printline()

- **Function Code examples :-**

(1) Without arguments and without return value
(2) without arguments and with return value.
(3) With arguments and with return value
(4) With arguments and without return value.

(4.) [with argument and with return value]

```
#include<stdio.h>
#include<conio.h>
void main()
{
int Sum(int a, int b)
  {
   return a+b;
  }
int a, b, c;
   a = 7;
   b = 8;
   c = Sum(a, b);
printf("The Sum is %d\n", c);
return 0;
getch();
}
```

input + output.

→ Declaration

→ Definition

2. With arguments and without return value.

```
#include<stdio.h>
#include<conio.h>
 void main()
{
 void printstar(int n)
   {
    while for(int i=0; i<n; i++)
       {
        printf("%c", '*');
       }

   }

 printstar(7);
 getch();
}
```

input but no output

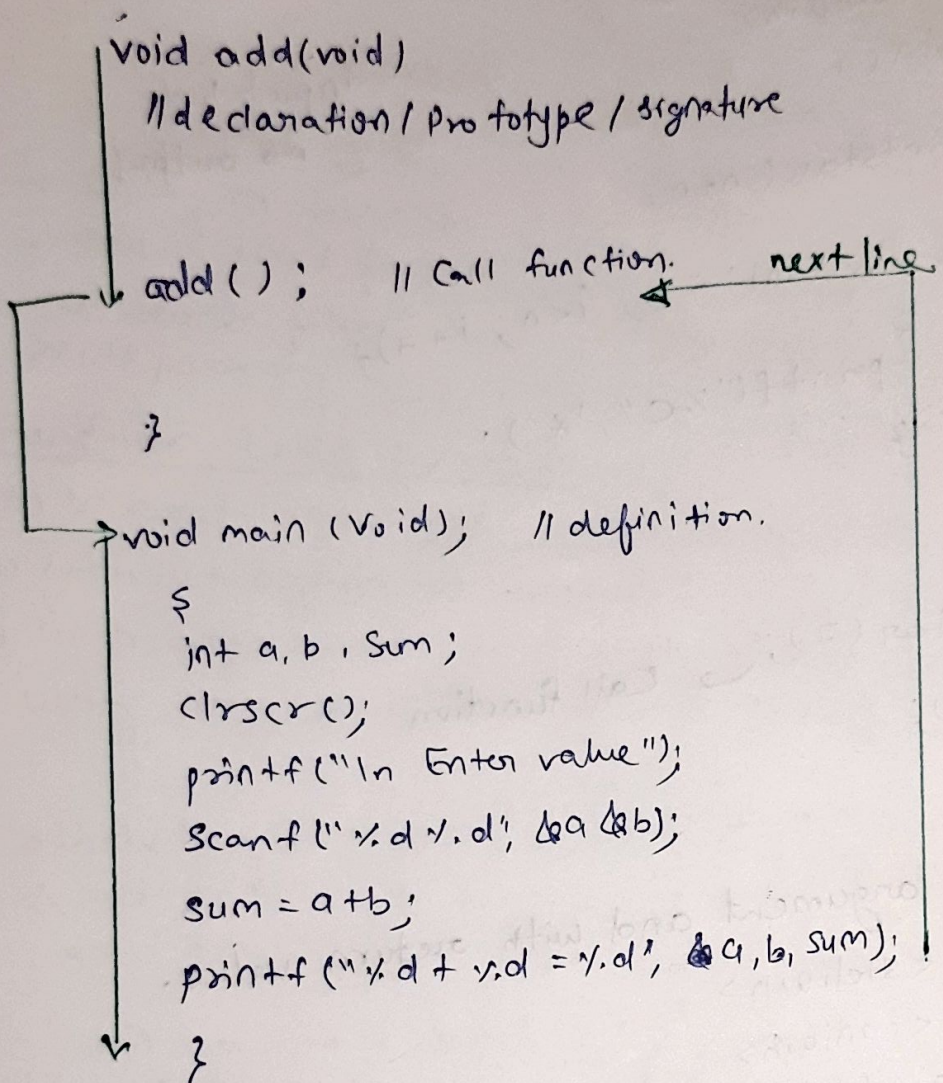
↳ Call function

3. Without argument and with return value.

```
#include<stelio.h>
#include<conio.h>
  void main()
  {int c;
  void takenumber()
       {
        int i;
        printf("Enter a number");
        scanf("%d", &i);
        return i;
       }

c= takenumber();
printf("The number entered is %d", c);
getch();
}
```

no input but output

4. Without argument and without return value :-




Note a function can call another function.

## • Array as parameter in function:-

```
main ( )
  {
  int array[ ] = {, , , , , , }
  int size = 5;
  use of array (array, size)

  }
```

```
void use of array
(int arr[ ], int n)
 {
for(i=0; i<n; i++).
 {
printf("%d", arr[i]);
}
}
```

## • Recursive function

When function call itself, then ~~this~~ it is called recursive function.

Ex

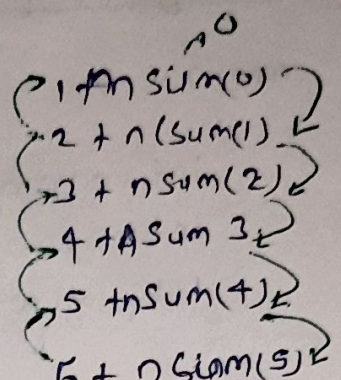
```
int factorial (int n)
    {
    if (n>=1)
      {
      return n * factorial (n-1);
      }
    else
      {
      return 1;
      }

main ( )
  {
  int factorial (int n)
  int fact (int n)
    printf("%d", factorial(6));
    printf("%d", fact(5));
```

```
int fact (int n)
  {
  fact = 1;
  int = 1;
for (i=1; i<=n; i++)
  {
  fact = fact * i;
  return fact;
  }
```

1
2 x factorial (1)
3 x factorial (2)
4 x factorial (3)
5 x factorial (4)
6 x factorial (5)

(Recursive function)

Ex
```
int nsum(int n)
{
 if (n==0)
 {
  return 0;
  int res = n + n(sum(n-1));
 }
  return res;
}
```




Ex
```
int fibonacci (int n)
{
 if (n==0)
 {
  return 0;
 }
 else if(n==1)
  {
   return (1);
  }
 else
  {
   return ( fibonacei (n-1) + fibonacei (n-2));
  }
```

# Pointer

→ Pointer is a variable which can stores another variable's address.

These are derived data type.

→ It can be used to return multiple values from a function.

→ Saving data storage space in memory.

→ It has own unique address.

| variable | value | Address |
|---|---|---|
| quantity | 179 | 5000 |
| p | 5000 | 5048 |

## Address of a variable -

* & → address of

p = &quantity;

// address of quantity
= 5000

p = 5000

## Declaring Pointers

data_type *pt_name;
↳ Pointer value.

Ex int *p;

float *p;

(here * is defrefrence Operator)

## Initialization of Pointer :-

Ex int quantity;
int *p; // Declaration.
p = &quantity; // address of quantity.

```
int x = 10;
int *p;
p = &x;
```

let

x [ 10 ] 6054 address of another variable

p [ 6054 ] 2024 (add of pointer.)

```
printf("%d", p);    // 6054
printf("%d", &p);   // 2024
printf("%d", *p);   // 10
printf("%d", x);    // 10
printf("%d", &x);   // 6054
```

for swaping

```
int a=10, b=5, temp;
int *pa, *pb
    pa = &a;
    pb = &b;
// Swaping
temp = *pa;
*pa = *pb;
*pb = temp;
```

[Call by refrence]

[Note pa of array will take base address directly so no & is used]

Ex

```
int arr[] = {10, 20, 30, 40, 50};
int *pa;
pa = arr;
for(i=0; i<5; i++)
{
printf("%d value %d at %d address", arr[i], &arr[i]);
printf("\n value %d at %d address", *pa, pa);
pa++;
}
```

[// arthematic operation are possible but only + & -]

# Structure

→ Structure is commonly refer as a user define data type. (Heterogenous)

→ Contains diverse information.

Use → storing employee info.

Syntax

```
{
    int id_no;
    char name[20];
    char desig[20];
    int salary;
};
```

members

```
struct Tag_name {
    datatype _ identifier
    datatype _ identifier
};
```

Program

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
structure employee
{
    int id_no;
    char name[20];
    char desig[20];
    int salary;
};
main()
{
employee emp1 // link with
printf(" Enter id_no : ");
scanf (" %d ", &emp1.id_no);
printf(" Enter your name :");
scanf (" %s", emp1.name);
```

keyword

struct tag

```
printf(" Enter designation :");
scanf(" %s", emp1.desig);
getch();
}
```

## Structure Pointer

```
#include <stdio.h>
struct person {
        int age;
        float weight;
    };
int main()
{
    struct person *personptr; person1;
    personptr = &person1;
    printf(" Enter age");
    scanf(" %d ", &personptr -> age);
    printf("\n Enter weight : ");
    fscanf(" %f ", &personptr -> weight);
    print f(" Displaying \n ");
    printf(" Age : %d \n ", personptr -> age);
    printf(" weight : %f ", personptr -> weight);
    return 0;
}
```

```
struct person {
        int age;
        float weight;
        char name[30];
        };
int main()
  {
   struct person *ptr;
    int i, n;
    printf("Enter no. of person: ");
    scanf("%d", &n);
 ptr = (struct person *) malloc (n * sizeof
                                 (struct (person));
for (i=0; i<n ; i++)
  {
  printf(" Enter first name & age ");
  Scanf("%s %d",(ptr+i)->name;
                 &(ptr+i)->age);
  }
for (i=0 ; i<n ; i++)
  printf("Name : %s\n %d",
          (ptr+i)->name,(ptr+i)->age);
  return 0;
}
```

- Using Call by value method

```
#include <stdio.h>
Struct car {
        char name[30];
        int price;
      };
void print_car_info(Struct car c)
    {
     printf("Name : %s", c.name);
     printf("\n Price : %d\n", c.price);
    }
int main()
    {
     Struct car c = {"Tata", 1021};
     print_car_info (c);
     return 0;
    }
```

Using Call by Refrence method

```
#include<stdio.h>
#include<conio.h>
    struct student {
                    char name[50]
                    int roll;
                    float marks;
                   };
void display ( struct student * student_obj)
{
 printf("Name : %s\n", student_obj->name);
 printf ("Roll : %d\n", student_obj -> roll );
 printf ("Marks : %f\n", student_obj->marks);
 }
int main()
    {
     Struct student st1 = {"Aman ", 19, 8.5};
     display (&st1);
     return 0;
    }
```

# Structure as Argument

```
Struct patient
{
int ID;
char Name[20];
};
void passbyvalue (struct patient p)
    {
    PID =120;
    printf("\n The patient ID %d "; P.ID);
    printf("\n The patient's name %s", P.Name);
    printf("\n The patient gender %c", p.gender);
    }
int main ()
  {
  Struct patient
  P1. ID = 101;
  Strcpy ( P1. Name, "Ganv");
  P1. gender = 'M';
  pass by value (P) // (call function)
  }
```

# Union

→ User defined Data type.

→ In structure, each member has its own memory storage location and for union, member uses a single shared memory location.

single shared memory ⇒ size of largest data member.
location

→ Structure can be store & used together.
(activate all)
but in Union at a time only one datatype work

Syntax

```
union emp1 {
    int emp_id;
    float emp_sal;
};
union un
```

for accessing

```
un.emp_id;
un.emp_sal;
```

→ access by operator (.)

→ Saves memory & make program efficient.

~~Structure Array~~:-

# FILE

→ Used to store data & info.
→ Used to read & access datatype datatime from hard disk
→ When a C program is terminated, data stored in RAM is lost, so we need to store it permanently.
→ easy to transfer data.

Types of File
- → Text file → data stored in text
- → Binary file → data stored in Binary.

Operation
(i) Creating
(ii) Opening
(iii) Closing
(iv) Reading
(v) Writing

Declaration with pointer of type FILE

Ex:
```
FILE *ptr = NULL;
```

for text file
"w" => write
"r" => read
"a" => appenel

for binary file
"wb" = write
"rb" = read
"ab" = appenel.

(w+ = read + write)

## (i) Creating or edition FILE

stdio.h → fopen() :- opening file

Syntax `ptr = fopen("fileopen", "mode");`
- fileopen → filename
- mode → purpose.

Ex: `fopen("iemp.txt", "w")`

fclose() → file close function.
(Stdio.h function)

## (ii) Reading a FILE:-

(fopen is must)

`fscanf(ptr, "%s", string)`

(Read mode)

↳ Reading function => (Stdio.h)

## Writing a FILE

use fprintf function => (stdio.h) (write mode)

Ex: `fprintf(ptr, "Name = %s\n", name);`

## Text file

```
void main()
{
FILE *fptr;
char name[20];
int age;
float salary;
int n, i;
fptr = fopen("emp.txt","w");
                //(write mode)
if(fptr = NULL)
  {
  printf("File does not exist");
  return 0;
  }
printf("Enter the name\n");
scanf("%s", name);
fprintf(fptr, "Name = %s\n", name);
printf("Enter age");
 scanf("%d", &age);
fprintf(fptr, "Age = %d\n", age);
printf("Enter salary");
scanf("%f", &salary);
fprintf(fptr, "Salary = %f\n", salary);
 fclose(fptr);
}
```

put file put string
fprintf => fputc()
fscanf => fgets()
w+ = read + writing (creates new file if not exist)
r+ = read + writing
a+ = open file for reading & writing. (from begening)

Ex

```
int main( )
{
FILE *fp, *fp1;
char ch;
clrscr();
fp = fopen("emp.text","r");
                    //(read mode)
fp1 = fopen("Newemp.text","w");
                    // (write mode)
if(fp==NULL || fp1==NULL)
{
printf("error in files");
}
ch = fgetc(fp);
while (ch != EOF)
                    //(EOF = end of the file)
{
fputc(ch, fp1)
ch = fgetc(fp);
}
fclose(fp);
fclose(fp1);
return 0;
}
```

Ex

```
- int main()
  {
  FILE *fptr;
  char ch;
  fptr = fopen("Emp.txt","r");
   if(fptr = NULL)
   {
   printf("file does not exist");
    return 0;
   }
  ch = fgetc(fptr);
  while (ch != EOF)
   {
    printf("%c", ch);
     ch = fgetc(fptr);
   }
    fclose(fptr);
    return 0;
  }
```

# modes & functions of file

r → read file
w → write for file
a → appending a file
r+ → read + write mode [replacing from beginning]
w+ → read + write [If file exist → truncates
file not exist → creates new one]
a+ → read + write [if file not exist → creates the file.
reading → start from beginning]

fputc
fputs } → Input (writing for file)

* c → char
* s → string

fgetc
fgets } → fscanf (reading for file

EOF ⇒ end of file => (stdio.h) constant return when it's fail.

Syntax

int x,
int fgets (const char *s, FILE *fp);

int fputc(char, FILE *ptr);

int fputs (const char *s, FILE *fp);

int fgetc (char, int x, FILE *ptr)

Program

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct student {
        int SNO;
        char Name[30];
        float marks
        };
void main()
    {
     struct student S[50];
        int i;
        FILE *fp;
        fp = fopen("student.txt", "w");
    }

for(i=0; i<10; i++)
    {
     printf("Enter Details of student %d", i++);
     printf("\n Student No ");
     scanf("%d", &S[i].SNO);
     printf("\n Student Name");
     gets(S[i].name);
     printf("\n Student marks");
     scanf("%f", &S[i].marks);
     fwrite(&S[i], sizeof S[i]; fp);
     }
     fclose(fp);
```

```
fp = fopen ("student.txt", "r");
for (i=0; i<10; i++)
{
printf("\n Details of %d student ", i++);
fread (&s[i], sizeof(s[i]), i, fp);
printf("\n student no = %d", s[i].sno);
printf("\n student name = %s", s[i].sname);
printf("\n student marks = %f", s[i].marks);
}
fclose (fp);
getch();
}
```

## * Command line Argument :-

→ used to supply parameters to the program when it is invoked.

→ Used when we need to control our program from the console.

→ ffmpeg.exe is a command line utility.

ex = git, brew, apt-get etc

argc → 1 when no arg. passed

argc ↓ number of arguments

argv ↳ array of all arguments.

```
#include <stdio.h>
int main (int argc, char* argv[]).
{
printf("argc is %d\n", argc);
if (argc >= 2)
{
printf("The arguments supplied are:\n");
for(int i=0; i<argc; i++)
{ printf("%s\t", argv[i]);
```

# ★★ Storage classes ★★

Storage classes in c are used to determine life time, visibilaty (scope), memory location & initial value of a variable.

Dynamic memory location ⇒ size of data structure can be changed during the runtime

(i) Scope ⇒ availability of variable in which range

(ii) Default initial value. → If we not initialize a variable specifically then what value it will take.

(iii) Lifetime ⇒ how much time it availability.

There are 4 types of storage classes in c

1. Automatic Storage class (Default class)
2. External Storage class
3. Static Storage class
4. Resistor Storage class.

| Storage Class | Storage Place | Default value | Scope (Visibility) | life |
|---|---|---|---|---|
| Auto | RAM | Garbage | local | life within function |
| Extern | RAM | 0 | Global | till the end of the main function |
| Static | RAM | 0 | Local | till the end of the main funct |
| Resistor | Resister | Garbage | LOCAL | With in the function. |

Ex:

```
int main ( )
{
int a;
char b;
float c;
printf("%d %c %f", a,b,c);
return 0;
}
```

(by default auto)

output
garbage.

Ex:

```
int main ()
{
int a =10, i;
printf("%d", ++a);
{
int a = 20;
for(i=0; i<3; i++)
{
printf("%d", a);
}
}
printf("%d", a);
}
```

within the function.

output

11
20
20
20
11

Ex: Static

```
static char ch;
static int i;
static float flt;
static char name[100]
void main().
{
printf("%d %s %i %f", ch, i, name, flt);
}
```

output

int = 0
float = 0.000
char → null
array.
Char = 0

Ex

```
int sum()
{
    int a=10;
    int b=24;
    printf("%d %d\n",a,b);
    a++;
    b++;
}
void main()
{
    for(i=0;i<3;i++)
    {
        sum();
    }
```

again again starting value

output

1024
1024
1024

(Auto)

Ex

```
int sum()
{
    static int a=10;
    static int b=24;
    printf("%d %d\n",a,b);
    a++;
    b++;
}
void main()
{
    int i;
    for(i=0;i<3;i++)
    {
        sum
    }
```

It will take its changed or modified value

output

1024
1125
1226

Ex
```
int main()
{
register int a;
printf("%d", a);
return 0;
}
```

output
garbage.

Ex
```
int main()
{
register int a = 0;
printf("%u", &a);
}
```

output
error.
because it will not store it in memory location.

Ex
```
int a;
int main()
{
extern int a;
printf("%d", a);
}
```

output = 0

Ex *
```
int main()
{
extern int x
printf("%d", x)
}
int x = 20;
```
output = 20

```
int main();
{
extern int a=10;
printf("%d", a)
}
```

output = error.

This will show error since we can not use extern & initialize at the same time.

* // Compiler will search here variable 'x' defined & initialize somewhere in the program or not.

# * ENUM *

Enumerated datatype (Enum) ⇒ user defined datatype.

→ Enum in c is a special kind of datatype defined by the user. It consist of constant integral or integer that are given names by a user.

Ex

```
#include <stdio.h>
enum year{Jan=1, feb, mar, april, may, june,
          July, Aug, sept, oct, Nov, Dec};

int main()
{
int i;
for (i=jan; i<=Dec; i++)
{printf("%d ",i);
 return 0;
}
}
```

output ⇒ 1 2 3 4 5 6 7 8 9 10 11 12