COMPUTER PROGRAMMING

FIRST YEAR NOTES

-AKSHANSH CHAUDHARY

Computer Programming Notes, First Edition

Presented by:	Akshansh Chaudhary Graduate of BITS Pilani, Dubai Campus Batch of 2011
Course content by:	Dr. Alamelu Mangai Then Faculty, BITS Pilani, Dubai Campus
Layout design by:	AC Creations © 2013

The course content was prepared during Spring, 2012.
More content available at: www.Akshansh.weebly.com

DISCLAIMER: While the document has attempted to make the information as accurate as possible, the information on this document is for personal and/or educational use only and is provided in good faith without any express or implied warranty. There is no guarantee given as to the accuracy or currency of any individual items. The document does not accept responsibility for any loss or damage occasioned by use of the information contained and acknowledges credit of author(s) where ever due. While the document makes every effort to ensure the availability and integrity of its resources, it cannot guarantee that these will always be available, and/or free of any defects, including viruses. Users should take this into account when accessing the resources. All access and use is at the risk of the user and owner reserves that right to control or deny access.

Information, notes, models, graph etc. provided about subjects, topics, units, courses and any other similar arrangements for course/paper, are an expression to facilitate ease of learning and dissemination of views/personal understanding and as such they are not to be taken as a firm offer or undertaking. The document reserves the right to discontinue or vary such subjects, topic, units, courses, or arrangements at any time without notice and to impose limitations on accessibility in any course.

☆ Problem solving with computers:

D
AM
DAM

1. Clearly define the problem.
2. Analyse the problem and formulate a method to solve it.
3. Describe the solⁿ in the form of an algorithm.
4. Draw a flow chart of the algo.
5. Write the computer program.
6. Compile & run the program.
7. Test the program.
8. Interpretⁿ of results.

A
F
W
C

R
T
I
RTI

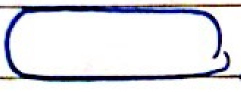
☆ Algorithm

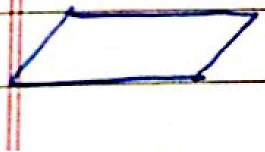
- a step-by-step description of the solⁿ
- can be written in a non-formal lang. & structure.

☆ Flow chart

- logical flow of solⁿ in a diagramatic form.
- provides a plan for writing comp. program.
- logical flow of an algo can be traced through the flow chart.

→ Std. flow chart symbols:


Start/Stop.


Input/Output

▭ Computation (Processing)

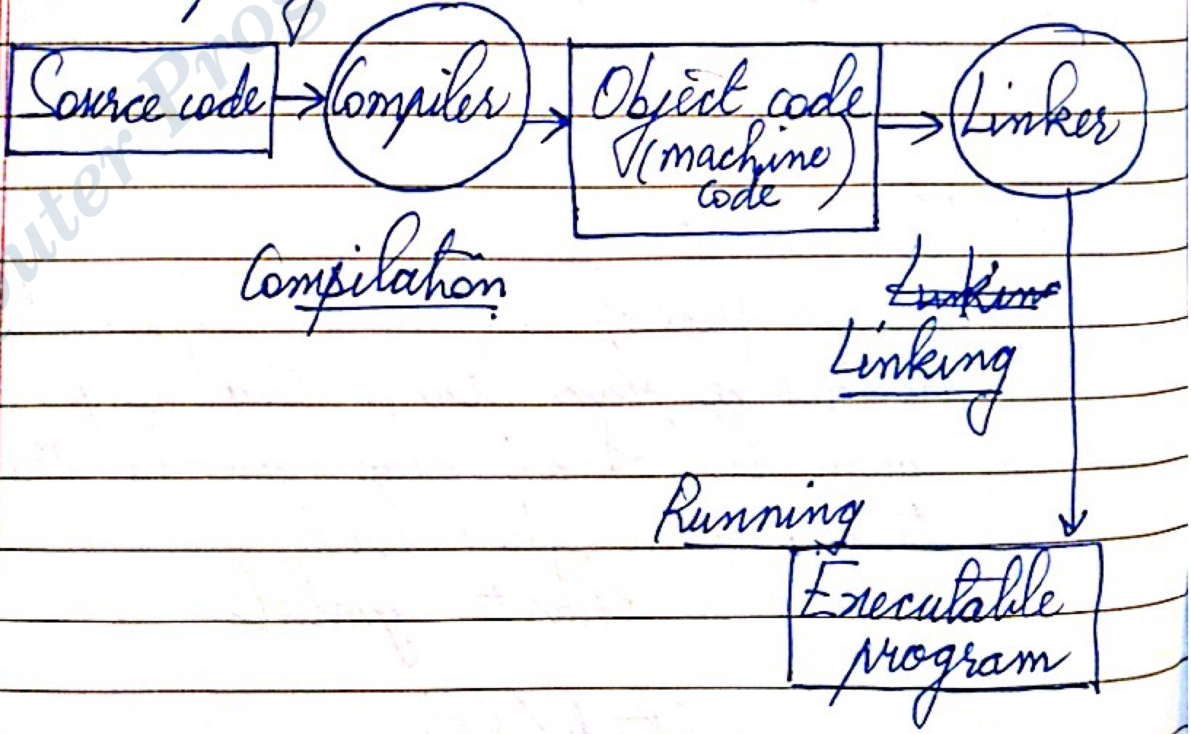
◇ Decision making (Decision box)

⬡ Repitition Structure (Looping statements)

↓ → Dir^n of control flow.

⫰ Connects the flow chart b/w pages.

★ Compiling a source code into executable m/c program:

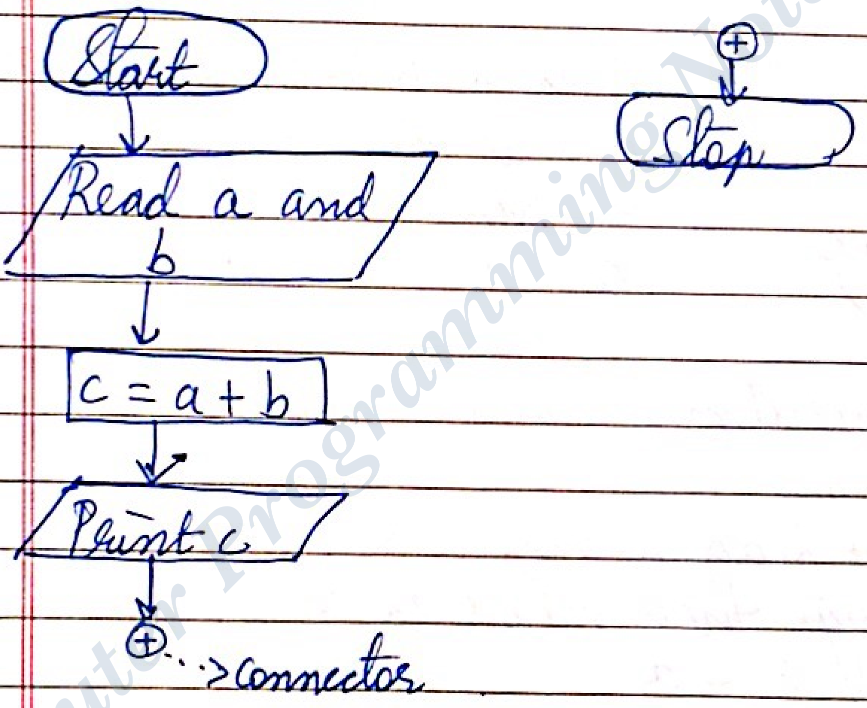


Problem Read 2 nos. & find their sum.

Algorithm

1. Start
2. Read 2 nos. in a & b.
3. Compute their sum as c = a+b.
4. Print c.
5. Stop.

Flow chart

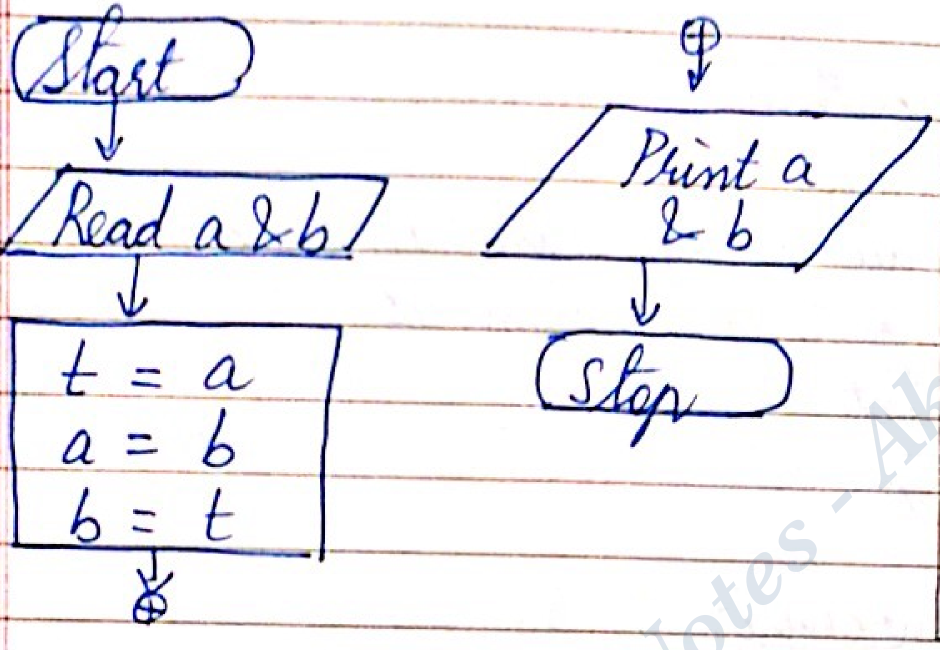


Problem: Read 2 nos. a & b and swap them.

Algorithm.

1 Start
2. Read 2 nos. in a & b
3. Assign ~~t~~ 'a' to a temp. variable 't'.
4. Assign 'b' to 'a'
5. Assign old value of 'a' in 't' to 'b'.
6. Print a&b, 7. Stop

Flow chart.



Aliter

* a = a+b
b = a-b
a = a-b

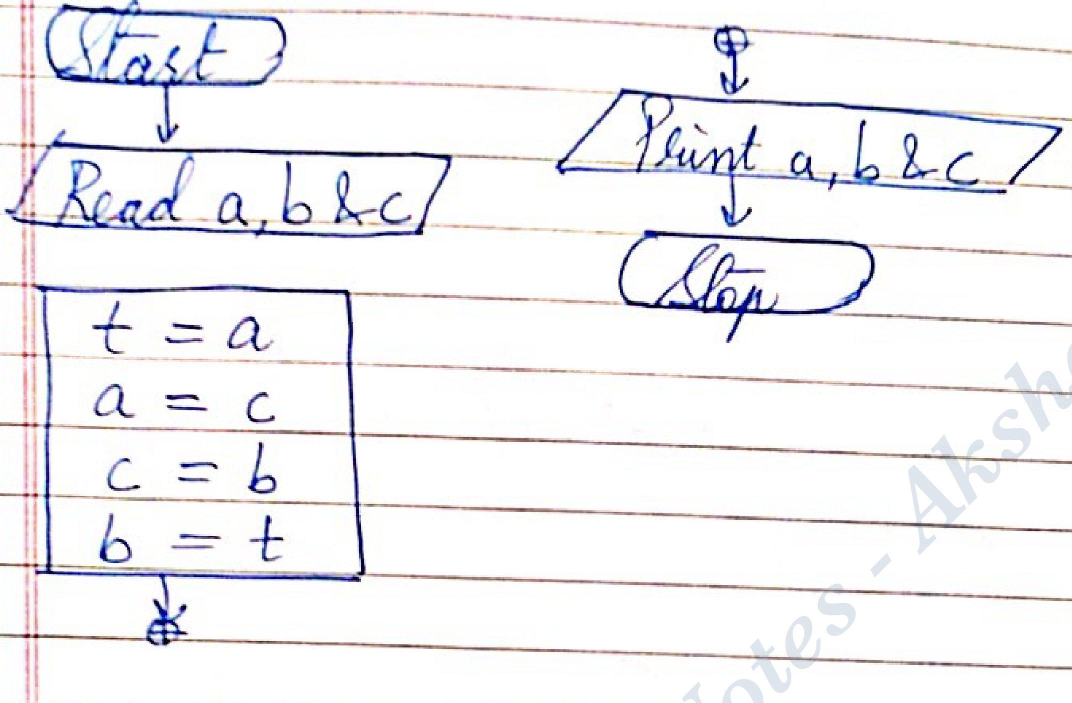
Program: Given 3 values a, b & c, rotate their values s.t b has value of a
c has value of b
a " " " c.

$$\begin{tikzcd} a \arrow[r] & b \arrow[r] & c \arrow[ll, bend left] \end{tikzcd}$$

Algorithm

1. Start
2. ~~Red~~ Read 2 nos. a & b
3. Assign these values to variables
   t = a
   a = c
   c = b
   b = t
4. Print a, b & c
5. Stop.

Flow chart

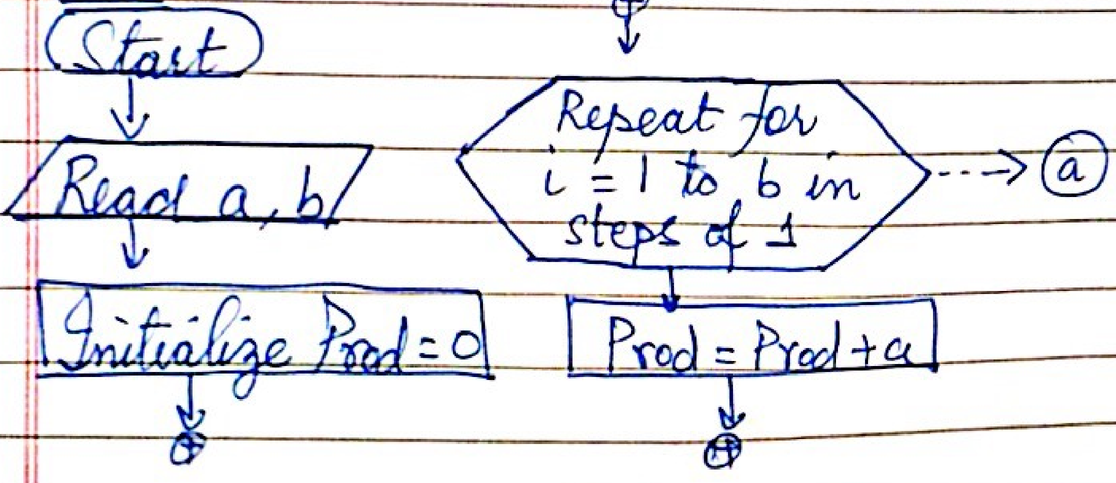


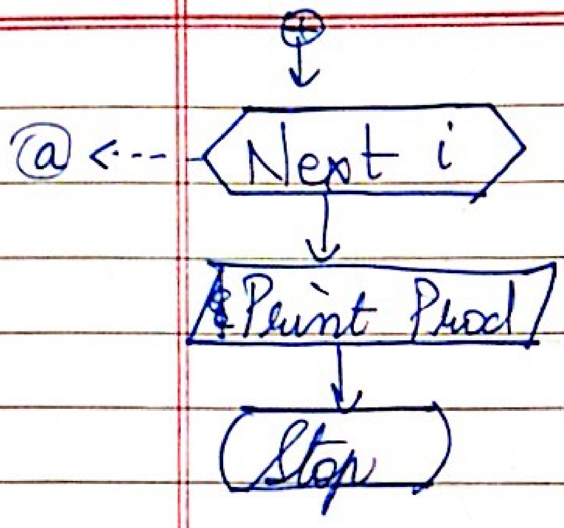
Program: Write the algorithm & draw the flow chart to find the product of 2 nos. a & b.

Algorithm

```
1. Start
2. Read 2 nos. a & b; 3. Initialize Prod = 0.
4. Repeat Step 5
   for i = 1 to b in increments of 1.
5. Prod = Prod + a.
6. Print Prod.
7. Stop.
```

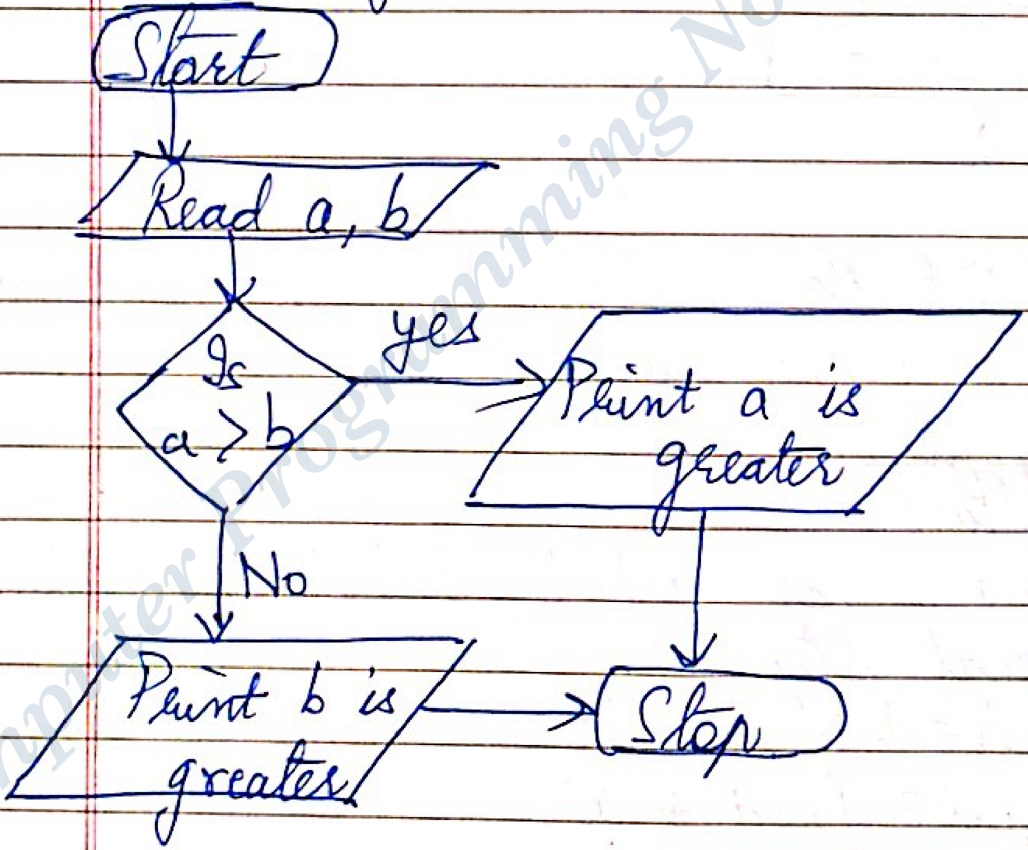
F.C





Program: To see greater of 2 nos.

FC



C

- case sensitive language.

* Data types

- a set of values & set of oper^{ns} on those values.

~~flat~~ float, double — real values
int — integer values.

- constraints → + or − sign
  → no comma (like ~~15,000~~)
- variables.

* Data type unit

- whole nos.
- −32767 through 32768 → including zero
- perform arithmetic oper^{ns} :

+ add^{n}, subtraction, multiplic^{n}, div^{n}, compar two integers.

* Data type DOUBLE

- real no.
- has → integral part & fractional part.
  → separated by decimal pt.
- double : eg : 3.14159, 0.0005 . . .

- very large & very small nos. scientific notⁿ
- arithmetic operⁿˢ & comparison.

eg: $1.23 \times 10^5 = 123000.0$

$= 1.23e5 = 1.23E5$.

1.23 → mantissa → real, → Z

5 → exponent → Z

* Double Constants

Valid	Invalid
3.14159	150.0
0.005	0·12345e
12345.0	15e-0.3 → Z
15.0e-04 (0.0015)	12.5e.3
2.345e2 (234.5)	34,500.99
1.15e-3 (0.00115)	
12e+5 (1200000.0)	

* Data type FLOAT

- to represent real nos.

* Data type CHAR :-

'A', '2', 'z', '+', ' ' → blank space

- compare
- arithmetic operⁿˢ.

classmate
Date 7/2/22
Page

* Data type VOID : empty value.

Chapter 2

2.1

÷ C language elemts :

- resembles everyday English.
- 2 parts
  - → Pre-processor directives
    - Cmds that give instructions to a C pre-processor.
    - Modifies the text before compil^n.
    - Begins with symbol → #.
  - → Main f^n.

* 2 common preprocessor directives.

1. #include.
2. # define.

1. #include: used for using library f^ns in a program. Some library f^ns are ~~sa~~ scanf(), printf(),

$\underbrace{\text{scanf(), printf()}}_{\text{Input/output } f^{ns}}$

⇓

Header file : stdio.h
(primary.extension)

Pow(), sqrt(), sin(), cos(), ...

⇓

Header file : math.h

for compiling any program under math.h header file, type "cc filename.c -lm" instead of "cc filename.c"

- Library: Useful info. ut ~~of~~ collection of symbols

* Library f^ns :-

- predefined
- grouped into header files ——— .h

1 I/O f^ns	Math f^n	String f^n
stdio.h	math.h	string.h
scanf()	sin()	strlen()
printf()	cos()	strcmp()
gets()	sqrt()	⋮
⋮	pow()	
	log()	
	⋮	

Format :: #include< header filename >
eg: #include <stdio.h>, #include <math.

- Users can also create library f^ns.

2. # define
You can give a name to a constant.
eg: # define pi 3.14.
Format :: # define symbolicname constant

* To make a comment in a program
/* . . . . . —— — — */

- non executable statements.

Second

* Function (main):

Syntax:

```
int main()    or    main()
{                   {
   fn body             fn body
}                   }
```

- fn body.

declarn part

executable stmt. part.

. which memory cells will be used in the program. Variable names tells which memory cells will be used.

eg: int main(void).

```
{
  printf("\n Hello \n");
  return(0);
}
```

* Reserved words : (Std. identefiers).

- has a special - fixed meaning.

ex: int void double return if for ....

• Keywords

- have fixed meaning
- written in lower case.

eg: auto for int if case
break goto float char default.

* Identifiers : Defined by users.
- names of variables, f^{ns}, arrays.

* Rules to be used for selecting identifier :-
1. 1st char : must be alphabet.
2. Consists of only letters, digits or underscore. (only underscore is allowed in special char.)
3. Only 1st 31 char, are significant.
4. Shouldn't be a keyword.
5. No white space, no limit on length.

* Stmt terminator :- " ; "

Variable names

Valid	Invalid
letter=1	1 letter
Letter_2	double
inches	int
cent	TWO * FOUR
Rate	joe's
rate	Letter-1
RATE	

CASE SENSITIVE { Rate, rate, RATE } will be treated as diff^t variable names

* General practice
- Symbolic constants : Upper case
- Rest of the program : Lower case

* Variable name should signify its usage in the program.

* For int variable, memory allocated = 2 byte
char 1 byte

* Variable declar^{n}

Syntax: data type variable name,..., . . ;

* Assignment Statement:

Assigning values to variables used in a program :-

(i) using an assignment stmt :-

Syntax: variable name = constant ;

↳ done at compil^{n} time

- Initializ^{n} :-

eg: int final-value = 10 ;

(ii) through input f^{ns} :-

↳ called as Runtime assignments done to variables

Syntax: scanf("controlstring", &variable1, &variable 2,...);

↳ address of input variable 1 & value of it

- Format / Control String:

→ The format in which the user should input values.

%d : integer

%f : float, double

%c : character

%s : string (no &)

like scanf("%d@%d", &a, &b);

Then, enter values as a@b

* Output function : printf

Syntax: printf("Format string", variable 1, var 2....);

→ only the value of variable (not the address)

* Escape sequence :

Character constts used for formatting the output. Begin with '\'.

Newline character '\n'

Horizontal tab '\t'.

Program :- Program to generate the following pattern of astericks :-

```
*
**
***
```

```
#include<stdio.h>

main()
clrscr();
{
printf(" *\n**\n***\n");

return(0);
}
```

★ **Operators :**
Symbols - mathematical / logical manipulations

★ **Types :**

A . Arithmetic operators
R . Relational "
L . Logical "
A . Assignment "
I . Increment & decrement
C . Conditional
B . Bitwise
S . Special

A A B C D
L
R S

★ **Expression :**
Combin^n of operands (constants / variables) by operators $\xrightarrow[\text{gives}]{\text{on evaluation,}}$ a single value.
Eg: 10+15
Result : 25 (any type, except VOID).

★ **Arithmetic operators**

★ Only with integer operand

Operator	Meaning
%	modulo division (remainder) (Z) integer

★ Binary : 2 operands for 1 operator.
★ Unary : 1 operand for 1 operator.
★ No operator for exponentiation — pow( )
★ Data type of operand = Data type of result.
★ When operators are in mixed mode — implicit type conversion is followed (Hierarchy).

* For modular operator, the sign is followed of 1st operand.

eg: $-14 \% 3 = -2$

$-14 \% -3 = -2$

$14 \% -3 = 2$.

* If one operand is integer & other is real

⇓

Mixed mode arithmetic

One operand	: $\mathbb{Z}$
Other operand	: $\mathbb{R}$
Result	: $\mathbb{R}$

eg: $15/10.0 = 15.0/10.0 = 1.5$.

But $15/10 = 1$.

eg②:

```
int a=5;
int b=2;
float c;
c = a/b.
  = 2.0
```

get actual value →

EXPLICIT TYPECASTING

```
int a=5;
int b=2;
float c;
c = (float) a/b
  = 5.0/2
  = 2.5
or → a/(float) b
  = 5/2.0
  = 2.5
```

Program:- Using arithmetic operators, read a 3 digit no. & find sum of its digits.

eg: 123

No	Digit = N%10	Sum = sum+digit	N=N/10
123	3	0+3 = 3	12
12	2	2+3 = 5	1
1	1	1+5 = 6	0

1/10 = 0

1 % 10 = 1

Program:-

```
#include <stdio.h>
main()
{
int num, d, sum=0; printf("Enter no");
scanf("%d", & num);
d = num % 10 ;
 sum = sum +d ;
 num=num/10 ;          } first digit

                10
d = ~~##~~ num % ~~d~~ ;
sum = sum+d ;          } second digit
num = num/10 ;

  d
~~num~~ = num % 10;
 sum = sum +d;         } 3rd digit

or
 sum = sum+ num ;

printf("%d", sum);
}
```

Program 2 :- WAP to read the radius of a circle and find its area, using symbolic constant.

```
#include<stdio.h>↵ #define PI 3.14
main ()
{
int r; ~~area~~ float area;
printf("Enter the radius of circle");
scanf("%d", &r);
~~PI~~ area = PI ~~3.14~~ * r * r;
printf("\nThe area of circle is %f", area);

return (0);
}.
```

Program 3 : WAP to read the area of a triangle given its three sides, a, b & c.

Hint, $s = \frac{a+b+c}{2}$

$area = \sqrt{s(s-a)(s-b)(s-c)}$

```
#include<math.h>
#include<stdio.h>.
main ()
{
int a,b,c, ~~s, area~~;
float s, area
printf("\nEnter the values of three sides
        of triangle a, b and c");
scanf("\n %d %d %d", &a,&b,&c);
 s = (a+b+c)/2.0;
 area = sqrt(s*(s-a)*(s-b)*(s-c));

printf("\n The area of triangle is %f",
        area);

return (0);
}
```

☆ Relational operators

↳ relational expression arrives from them.

Priority : arithmetic ~~expression~~ operators (ae) > Rel^{nal} ~~expression~~ operators

Types of operator

- <
- <=
- >
- >=
- ==
- !=

Relational expression

ae-1 rel^{nal} operator ae-2

eg: 4.5 <= 10

(a+b) == (c+d)

→ arithmetic expression

Operator	Its complement
>	<=
<	>=
==	!=

* Logical Operators

↳ combines 2 or more arithmetic expression (ae.)

Operator	Meaning
&&	logical AND
||	logical OR
!	logical NOT

If ((age > 55) && (salary < 1000))

* Shorthand assignment operator

v op = exp.   v: variable

op: oper^n : linear, arithm

exp: expression

eg: a = a+10. ≡ a += 10

x = x + (y+2) ≡ x += y+2

a = a*(n+1) ≡ a *= n+1

* Increment and decrement operator.

- ++ and --
- unary, have only variables as operands

m++ ≡ m=m+1 ; m-- ≡ m=m-1.

* Prefix and Postfix form

(Pre-increment & post increment)

Pre	Post	Ex.	ex
++m	m++	a=5	a=5
--m	m--	b=a++	b=++a

		Result
		a=6
		b=5
			Result
			a=6
			b=6

first value of a is assigned to b, then incremented

* Conditional Operator

Symbol : ? :

Syntax : $\underbrace{exp1}_{\text{rel}^{\text{nat}}\text{/logical}}\ ?\ \underbrace{exp2}_{\text{evaluated if exp1 is true}}\ :\ \underbrace{exp3}_{\text{evaluated if exp1 is false}}$;

eg : a = 10, b = 5

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

↳ used to compare 2 nos.

* Evaluation of expressions :

Using of Precedence (Priority) & associativity

- Operator (arithmetic) Priority

* / %	high
+ −	low

- Operators on same level have same priority. So, if they occur together, use associativity :- LEFT TO RIGHT.

eg:- $\underbrace{\underbrace{\underbrace{a*b}_{①}/c}_{②}-d}_{③}$

- Parenthesis : highest priority.

eg: $\underbrace{a*\underbrace{(\underbrace{(b/c)}_{1}-d)}_{2}}_{3}$

* In case of nesting of parenthesis, the innermost parenthesis is evaluated first.

* Rules for evaluating expressions :

(a) Parenthesis Rule :

(b) Operator Precedance rule

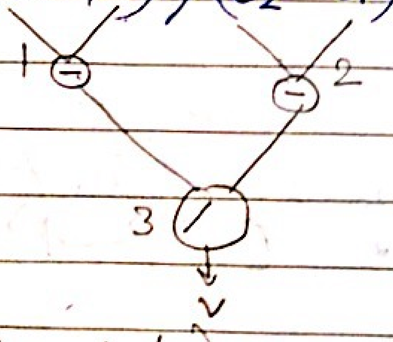
Operator	Precedance
• f^n ~~calls~~ calls (eg sqrt)	highest
• !, +, -, &, (unary), size of, (type) — type casting	
• * / %	
• + −	
• < <= >= >	
• = !=	
• &&	
• ||	
• =	Lowest

(c) Associativity rule

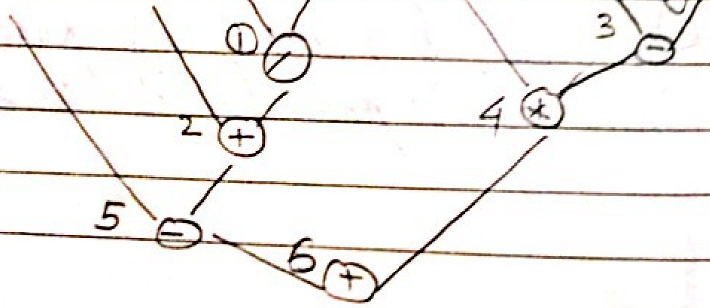
Unary operator : Right to left

Binary operator : left to right.

eg: $V = (p_2 - p_1)/(t_2 - t_1)$



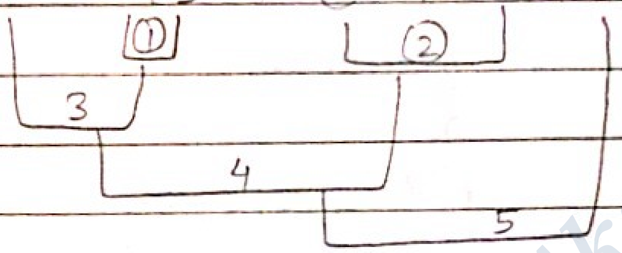
eg ② $z - (a + b/2) + w * - y$



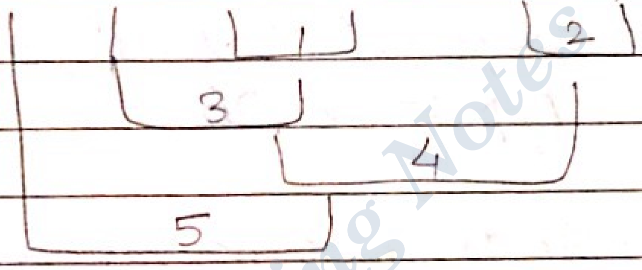
ex :- $x = a - b/3 + c * 2 - 1$

if $a = 9,\ b = 12,\ c = 3$

$x = a - b/3 + c * 2 - 1 = 10$.



ex :- $9 - 12/(3+3) * (2-1) = \boxed{7}$



* BITWISE OPERATORS :

- manipulates data at bit level
- testing bits, shifting them right/left.
- may not be applied to float or double

Operator	Meaning
&	bitwise AND
|	" OR
^	" exclusive OR
<<	shift left
>>	" right.

gives output as 1 only if the input operands are different

* Special operators.

(2) Comma operator :

- Link the related expr^{ns} together.
- Left → Right ; result is right most expr^{n}.

eg : value = (x=10, y=5, x+5);

(b) Size of operator :

```
eg:- int sum;
        m = sizeof (sum)        | m = 2
        n = size of(float)      | n = 4
eg: main()
     {
     int a,b,c,d
     a=15; b=10; c= ++a-b;
     d = b++ +a;
     printf(" %d %d", c,d);
     O/p :- 6 26
```

Program:- WAP to check whether a given no. is odd or even.

```
#include<stdio.h>
main()
{
int a, b ;
printf("\n Enter the no. to check");
scanf("\n%d", &a);
b = a%2
  (b = = 0) ? printf(" No. is even"); :
    printf("\n No. is odd");
return(0);
```

Check

Program:- WAP to find largest of 2 given nos.

```
#include<stdio.h>
main ()
{
```

a,b

(a>b)? printf("_ _"): printf(

```
int a, b ;
printf(" \nEnter the 2 nos to compare ");
scanf("\n%d %d", &a, &b);
 (a>b)? printf("\n a is greater ):
  printf("\n b is greater") ;
return(0);
}.
```

Program: WAP to read 3 integer nos. and find largest of them using conditional operators.

```
 #include <stdio.h>
main()
{
int a, b, c, d;
printf("\nEnter the 3 nos. to compare");
scanf("%d %d %d", &a, &b, &c);
```

```
(a>b)? ((b>c)? printf("\na>b>c"): printf
      ("\n a>c>b")): ((a>c)? printf("\nb>a>c")
      : printf("\n b>c>a ."));

return(0);
}.
```

✱ Formatting nos. in Program Output :-

Formatting int :-

%wd

w :- field width ; right justified

Value	Format	Displayed Outpu
234	%4d	␢ 234
234	%5d	␢␢234
234	%6d	␢␢␢234
234	%1d	234
-234	%4d	-234
-234	%5d	␢-234
-234	%6d	␢␢-234
-234	%2d	-234

Formatting double

% w.df

w: field width , d: no. ~~of~~ /... of digit / output column after the decimal point, sa up to d pos^ns.

- Output is right justified.

ex :- %6.2f

Value of x	Displayed output
-99.42	-99.42
.123	␢␢0.12
-9.536	␢-9.54
-25.554	-25.55

Value	Format	Output
3.14159	%.5.2f	ƀ3.14
3.14159	%.3.2f	~~ƀ6~~3.14
3.14159	%.5.3f	3.141
.1234	%.4.2f	0.12
-0.006	%.8.3f	ƀƀ-0.006

☆ int main(void)
→ after running the program in main, the f^n goes to OS to return a value zero.
So, we type return(0); in the end.

Chapter - 7

* Representⁿ & conversion of numeric types.

- Data is stored internally as binary strings.
- Binary string of int 13 is diffᵗ from float 13.0

type int	type double	
binary no.	mantissa	exponent

```
# include<stdio.h>
# include< limits.h> /*INT_MAX*/
#include <float.h> /*DBL_MIN, DBL_M*/
main()
 {
printf("Range of +ve values of type int :...
      ...%d\n", INT_MAX);
printf("Range of +ve values of type double
     : %e.......  %e\n", DBL_MIN,
        DBL_MAX);
```

%e : Meant for a real no. to be printed in exponent form

* signed : +ve as well as -ve values

* unsigned : only +ve values.

* The for stmt. is meant for int. variables.
eg: in real variables,

```
for (trial = 0.0; trial < 10.0; trial = trial + 0.1)
  {
     —
  }.
```

Some compilers run → 100 times,
others → 101 times. So, avoid.

* Automatic conversion of data type occurs in mixed mode arithmetic.
↳ Conversion done from lower level of hierarchy to higher level.

* Representn & conversion of type char.
- char – 'A', '+' 'a' '$'
- char values can be compared using relnal operator
  = =; ! =; <; <=; >; >=

ex:- char next-letter; /* char var */
        next-letter = 'A';
        if (next-letter < 'Z')

* Character values/codes (ASCII) are from 48 to 57 for 0 to 9

Date 19/2/12
Page

Chapter - 4.

Control Structures

↳ decide order of program execution

sequential (by default)
Writing the program in a sequence/order & executing it

selective
Execute stmts & by pass other stmts based on test cond^{ns}.

Types :-
↳ single alternative

① Simple if

```
If (cond^n)
    stmt ;
```

② If ... else

```
if (cond^n)
    ~~else {~~ stmt;
else
        stmt;
```

③ ~~Else~~ Nested ... if stmt

```
~~Else~~ { If (cond^n)
            stmt
Else
    {If (cond^n)
        stmt
    Else
        stmt
    }
}.
```

repetitive (or Iterative)
Multiple alternative nested if i.e If-else-if statement

```
If (cond^n)
    stmt;
else if (cond^n)
    stmt;
    :
    :
else if (cond^n)
    stmt;
else
    stmt;
```

↳ executed if all above cond^{ns} are false.

Program :- Read 3 nos. a, b & c to find largest of them

```
#include<stdio.h>
main()
{
int a,b,c;
printf("\nEnter 3 nos. a, b & c")
scanf("%d %d %d",&a,&b,&c);

if (a>b)
  {
   if (a>c)
       printf("\n a is largest \n");
   else
       printf("\n c is largest \n");
  }
else
  {
    if (b>c)
        printf("\n b is largest \n");
    else
        printf("\n c is largest \n");
  }

return(0);
}
```

Aliter

```
    if ((a>b)&&(a>c))
        printf("\n a");
    else if ((b>a)&& (b>c))
        printf("\n b");
    else
        printf("C");
```

—x—

* GOTO STATEMENTS

: unconditional transfer of program control

Syntax :-
```
goto label;
  - - -
  - - - -
                         } Forward jump
label :                  } - bypass some
     statement;          }      statements

label :
     statement           } backward
  - - -                  }     jump.
  - - - -                }
   goto label;           } - forms a loop
```

Program :- WAP to read the marks of a student in 8n subjects & find the total and average.

```
#include <stdio.h>
main()
{
int n, sum=0, m, count=0;
float avg;
```

```
    printf("\n Enter the no. of subjects");
    scanf("\n %d", & n);
l1: printf("\n Enter the marks in a subject");
    scanf("%d", & m);
    Sum = Sum + mark;
    count = count ++ ; (or count = count + 1 ;)
    if (Count <= n)
            goto l1;
    else
        printf("\nThe total marks are %d", sum);
                          typecasting operator
    avg = (float)sum/n;

    printf("\n The average is %f\n");

    return(0);
    }
```

★ Number System & representations

- Binary system : Base - 2 (0, 1)
- Decimal system : Base - 10 (0 to 9)
- Octal System : Base - 8 (0 to 7)
- Hexadecimal Sys. : (0 to 9) & (A to F)

A - 10 D - 13 : Base - 16.
B - 11 E - 14
C - 12 F - 15

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	-	8
9	1001	-	9
10	1010	-	A
11	1011	-	B
12	1100	-	C
13	1101	-	D
14	1110	-	E
15	1111	-	F

↓

To remember, write them as

8(0's)↓ 4(0's) 2(0's) 1(0)
2(1's) 1(1)
8(1's)↓ 4(1's) 2(0's) 1(0)
4(0's) 2 1(1)

* Convert decimal to binary

eg:- $(41)_{10} = (?)_2$

```
2 | 41
2   20 — 1
2   10 — 0
2    5 — 0
2    2 — 1
     1 — 0
```

Ans → $(101001)_2$

eg

```
2 | 45
2   22 - 1
2   11 - 0
2    5 - 1
2    2 - 1
     1 - 0
```

$(45)_{10} = (101101)_2$

* Decimal to octal

eg $(41)_{10} = (?)_8$

```
8 | 41
    5 — 1
```

Ans = $(51)_8$

eg:

```
8 | 324
8   40 — 4
     5 — 0
```

Ans = $(504)_8$.

* Decimal to hexademical

eg) $(41)_{10} = (?)_{16}$

$$\begin{array}{r|l} 16 & 41 \\ \hline & 2 - 9 \end{array} \quad = (29)_{16}$$

eg) $(45)_{10} = (?)_{16}$

$$= \begin{array}{r|l} 16 & 45 \\ \hline & 2 - 13 \end{array} \quad \text{Ans} = (2D)_{16}$$

* Binary to decimal

eg $(10110)_2 = (?)_{10}$

$$2^0 \times 0 + 2^1 \times 1 + 2^2 \times 1 + 2^3 \times 0 + 2^4 \times 1$$

$$= 2 + 4 + 16$$

Ans $= (22)_{10}$

* Octal to decimal

(eg) $(172)_8 = (?)_{10}$

$$2 \times 8^0 + 7 \times 8^1 + 1 \times 8^2$$

$$= 64 + 2 + 56$$

Ans $= (122)_{10}$

* Hexadecimal to decimal

256
×16
4096
×7
28672

eg $(7A1E)_{16} = (?)_{10}$

$E \times 16^0 + 1 \times 16^1 + A \times 16^2 + 7 \times 16^3$

$= 14 \times 16^0 + 1 \times 16^1 + 10 \times 16^2 + 4096 \times 7$

$= 14 + 16 + 2560 + 28672$

Ans $= (31262)_{10}$

28672
+2560
+30
31262

* Binary to hexadecimal

eg $\underbrace{1011}_{B}\,\underbrace{0101}_{5}$ (Group bits into 4 starting from right. If bits are less, put zeroes to make group.)

Ans $(B5)_{16}$

eg :- 11 0101

$= \underbrace{0011}_{3}\ \underbrace{0101}_{5}$ Ans $= (35)_{16}$

eg: $\underbrace{0110}_{6}\,\underbrace{1011}_{B}\,\underbrace{1000}_{8}\,\underbrace{1100}_{C}$

Ans $= (6B8C)_{16}$

* hexadecimal to binary

(write 4 bit binary equivalent of each hexadecimal)

eg) $(374F)_{16} = (?)_2$

3	7	4	F
0011	0111	0100	1111

Ans → $(0011\ 0111\ 0100\ 1111)_2$

* Converting fractions : decimal to binary

eg $(0.182)_{10} = (?)_2$

			Integer
0.182×2	=	0.364	0
0.364×2	=	0.728	0
0.728×2	=	1.456	1
0.456×2	=	0.912	0
0.912×2	=	1.824	1
0.824×2	=	1.648	1
0.648×2	=	1.296	1

Ans :- $(.0010111)_2$ ~~0.0010~~

- Proceed till the fractional part becomes zero or till the no. of digits got gives the required accuracy.

eg $(0.125)_{10} = (?)_2$

0.125×2	=	0.25	0
0.25×2	=	0.5	0
0.5×2	=	1.0	1
0×2	=	0.0.	

⇒ Ans = $(.001)_2$.

eg) $(41.125)_{10} = (?)_2$.

Convert integer part & fractional part separat

$(0.125)_{10} = (.001)_2$.

* Fractional binary to decimal

eg: $(.001)_2$
$\times 2^{-1}$
$\times 2^{-2}$
$\times 2^{-3}$

$$
\begin{array}{l|ll}
2 & 41 & \\
2 & 20 & 1 \\
2 & 10 & 0 \\
2 & 5 & 0 \\
2 & 2 & 1 \\
 & 1 & 0
\end{array}
$$

Ans $= (101001)_2$

$\Rightarrow (41.125)_{10} = (101001.001)_2$

* Decimal fraction to octal

ex:- $(0.125)_8 = (?)_8$

$0.125 \times 8 = 1.0$ ——— 1

$\therefore (0.125)_{10} = (0.1)_8$

* Decimal fraction to hexadecimal

eg $(0.9)_{10} = (?)_{16}$

$0.9 \times 16 = 14.4$	14	E
$0.4 \times 16 = 6.4$	6	6
$0.4 \times 16 = 6.4$	6	6

$\Rightarrow (0.9)_{10} = (.E66)_{16}$

* Binary fraction to decimal

eg: $(0.11)_2 = (?)_{10}$

$= 1 \times 2^{-1} + 1 \times 2^{-2}$

$= \frac{1}{2} + \frac{1}{4} = \frac{3}{4} = (0.75)_{10}$

eg $(45.75)_{10} = (\ .\ )_2$

```
2|45            (101101)₂
2 22  1
2 11  0         .75 × 2 = 1.5    1
2  5  1         .5 × 2  = 1.0    1   (0.11)₂
2  2  1         Ans:-
   1  0         (101101.11)₂
```

$1\times2^0 + 0\times2^1 + 1\times2^2 + 1\times2^3 + 0\times2^4 + 1\times2^5$

$1 + 4 + 8 + 32 = 45$

$1\times2^{-1} + 1\times2^{-2} = \frac{1}{2} + \frac{1}{4} = 0.75$

$\Rightarrow (45.75)_{10}$

* How to represent nos. in a computer?

- Numbers : signed, unsigned, integers, floating point, rational . . .
- Text : characters, strings
- Images : pixels, colors
- Sound : multimedia data
- logical : true, false

1) Unsigned integers :

- For representing memory addresses.
- weighted pos^{nal} not^{n} – like decimal nos.

- LSB : Least significant Bit
- MSB : Most significant Bit

eg 3 2 9 → 10^2, 10^1, 10^0
= 300 + 20 + 9 = 329

MSB ← 1 0 1 → LSB → 2^2, 2^1, 2^0
= 1 + 4 = 5.

★ An n-bit unsigned integer represents 2^n values : from 0 to $2^n - 1$

eg: n = 3, values = $2^3 = 8$
range = 0 to 7.

unsigned integers	= decimal equivalents
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

★ Binary addition on unsigned integers
- add from right to left, propagating carry

```
ex:-  1 0 0 1 0
    +   1 0 0 1
    -----------
      1 1 0 1 1

ex:-  1 0 0 1 0
    +   1 0 1 1
    -----------
      1 1 1 0 1
```

Truth table

a	b	sum	carry
1	0	1	0
0	1	1	0
0	0	0	0
1	1	0	1

★ 2) Signed Integers
- +ve & -ve Z.
- with n-bits, 2^n distinct values.
- assign half to ~~posn~~ +ve integers (+1) through 2^{n-1} -
- assign half to -ve integers $-(2^{n-1}-1)$ through -

- one value for 0, one not used.

eg: $n = 5$, values : $2^5 = 32$

+ve Z_s = 1 through $2^4 - 1$
= 1 through 15. 15

-ve Z_s = -15 through -1 15

one pattern = 0 1

one not used 1

$2^5 = 32$

* +ve Z_s :-

· Just like their unsigned equivalents.

· Zero is in MSB.

eg:- $\textcircled{0}0101 = 5$

* -ve Z_s

(a) Sign magnitude form :- · MSB = 1
· other bits same as its unsigned equivalent

ex:- $\textcircled{1}\overbrace{0101}^{5 \text{ bits}} = -5$, $\underbrace{10000101}_{8 \text{ bits}} = -5$

(b) One's complement

(i) Write the represent^n of +ve equivalent

(ii) flip each bit.

eg:- 5 bit represent^n :- 5 = 00101

flip the bits (0 to 1 & vice versa) :- 11010

*(c) 2's complement

· 1's complement + 1.

· Taking 2's complement is same as negating the number.

2's complement of X.

i.e $\boxed{X + (-X) = 0}$

original X

eg:- 2's complement of 5 → 00101

+ve equivalent

1's complement → 11010
+ 1

11011 → 2's complement

To prove: 2's complement + Original value = 0

```
    00101   (5)
 +  11011   (-5)
  ① 00000   (0)
```

carry → ignore

* Binary Subtraction

ex:- $7_{10} - 5_{10} = 7_{10} + (-5)_{10}$

(Using 4 bits) → 2's complement of 5.

7 in binary 0111_2

-5 in binary 1011_2

```
0101   original
1010   1's
  +1
1011   2's
```

```
   7      0111
+ (-5)    1011
        ① 0010
```

→ Ignore the carry.

eg. Add -25_{10} to 18_{10} using 6 bit represent^n

```
2|25       011001        -25₁₀ = 100111.
2 12  1  → 100110
2  6  0       +1
2  3  0    100111
   1  1    → 25.
```

2	18	
2	9	0
2	4	1
2	2	0
	1	0

$18_{10} = (10010)_2$

So, $18_{10} = (010010)_2$

~~0~~

$$\begin{aligned}(-25)_{10} &= 100111\\ (18)_{10} &= +010010\\ \hline &\ \ \ 111001 \text{ Ans.}\end{aligned}$$

Check:-

Find decimal equivalent

$(1\times(-32)_{10}) + (1\times 16_{10}) + (1\times 8_{10})$

∵ MSB = 1
So, -ve no.

$+ (1\times 2^0_{10}) = -7_{10}$.

Program: Read the coefficients a, b & c of a quadratic eqn
$ax^2+bx+c=0$
& find its roots.

```
#include<stdio.h>
main()
{
float a, b, c, d, rp, ip, r1, r2;
 int i;
printf("\n Enter the values of a, b & c
       for the equation ax^2+bx+c=0\n")
scanf("%f %f %f \n");
d = sq(b) - 4*a*c;

if ( d ==0)

      r1 = (-b)/2*a ;
      r2 = (-b)/2*a ;
      printf("\n The roots are equal and
      equal to %f and %f \n",r1,r2);

else if (d>0)
         r1 = (-b + sqrt(d))/2*a;
         r2 = (-b - sqrt(d))/2*a;
         printf(" \n The roots are real and
         unequal and are %f and %f",
         r1,r2);

else if (d<0)
         rp = (-b)/2*a;
          ip = (sqrt(abs(d)))/2*a ;
         r1 = rp + i(ip);
         r2 = rp - i(ip);
         printf("\n The roots are imaginary
          and equal to %f & %f \n", r1,
          r2);

return(0);

}
```

✱ Switch Statement

Syntax:

```
switch (controlling expression)
       {
       Case label set 1;
                 statements ;
                 break;
       Case label set 2:
                 statements ;
                 break;
       default :
              statements ;
       }
```

int or char

int or char

any number of them can be used

optional

Break: It is reqd for making only that stmt. to execute.

eg :-

```
   switch (watts)
      {
       case 25 :
              life = 2500;
              break;
       Case 40 :
       Case 60 :
              life = 1000;
              break;
      }case 75:
       Case 100 :
              life = 750;
       default :break;
   }          life = 0;
```

multiple labels can be given. Each label must have its own case.

Program:- Read a no. from 1 to 10 & print its name.

```
# include <stdio.h>
main ()
 {
 int n;
 printf ("\n Enter the number from 1 to
 10 to print its name \n");
 scanf ("%d", &n);
  switch (n)
      {
          case 1 :
                  printf("\n One");
                  break;
          Case 2 :
                  printf ("\n Two");
                   break;
          Case 3:
                 printf(" \n Three");
           ©     break;
          Case 4 :
                 printf(" \n Four");
                 break;
          Case 5:
                 Printf (" \n Five");
                 break;
              :
              :
          default :
                 printf(" \n Number entered is
          } break; more than 10 \n");
}
```

★ Char values in case & switch

eg
```
main ()
 {
 char c='x';
  switch (c)
   {
    case 'v':
       printf("\n I am in case v \n");
       break;
    case 'a':
       printf("\n I am in case a \n");
       break;
    case 'x':
      printf("\n I am in case x \n");
      break;
    default:
      printf("\n I am in default \n");
    }
  }
```

★ Note: We can mix integer & character values in diff^t^ cases of a switch.

★ Mix int & char values

```
main ()
 {
  int c=3;
  switch (c)
   {
    case 'v':
          printf("\n case v");
          break;
     case 3 :
          printf("\n I am in case 3");
          break;
     default:
          printf("Default case");
     }
}
```

Program: WAP using if & goto to find sum of the series :-

$1^2 + 2^2 + 3^2 + \ldots + 10^2$.

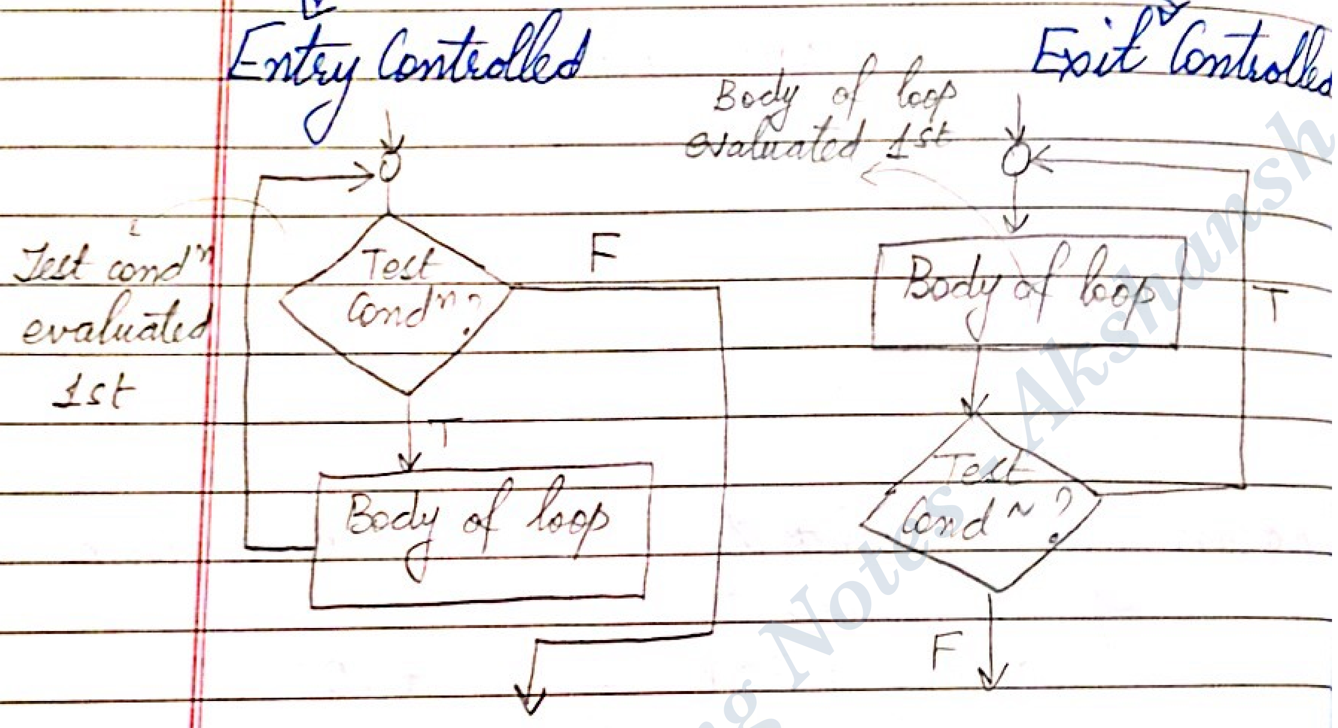
```
#include<stdio.h>
main()
{
int n=1, sum=0;

    ab:

    sum = sum + n*n;
    n++;
    If (t<=10)
          goto ab;
    printf("\n Sum = %d", sum);
}
```

*Program can be written with counter also.

* Looping / Repititive Control Structures



* Looping Statements

- WHILE stmt.
- DO-WHILE
- FOR

Q1 While stmt.

Syntax: While (test cond^n)

```
{
    body of the loop ;
}
```

- While : entry controlled looping structure.

Way * initialize counter variable ;

while (test cond^n or counter variable) ; not there

```
{
    ~~test cond~~ body of loop ;
        update counter ;
}
```

Program:- WAP to find sum of series 1+2+3+ - - - +10 using while loop.

```
# include< stdio.h>
 main()
 {
 int  n=1, sum=0;

 while ( n <=10)
 {
   Sum= Sum+ n;
   n++
 }
 printf("\n The sum of 1+2+...+10= %d"
   , sum);
}
```

eg:- WAP to find sum of series 2-4+6-8+10.

```
# include <stdio.h>.
# include< math.h>
main()
{ int  n=2, Sum= 0, i=0;
  while ( n <=10)
   {
     Sum = Sum + pow(-1, i)* n;
     n = n+2;

       i += 1;
   }
   printf("\n Sum is %d", Sum);
}
```

§ 2. DO WHILE stmt. (exit controlled looping stmt) f

SYNTAX :-

```
do
{
    body of loop;
}
while (test condn) ;
```

§ 3. FOR stmt.

Syntax:

```
for (initializn ; test condn ; update)
{
    body of loop ;
}.
```

; not there

ex:- WAP to find sum 1+2+3+ - - - +10 using for stmt.

```
int sum = 0;
for (i=1, i<=10, i++)
{
    Sum = sum + i ;
}
printf ("\n Sum is %d", sum);
```

{ } are not reqd in case of single stmt.

using do while stmt.

```
sum = 0 ; i = 1 ;
do
{
    sum = sum + i ;
    i = i + 1 ;
}
while ( i <= 10 )
```

Program - WAP to read a no. & check whether its a Palindrome or not.

```
# include < stdio. h>
 main ()
 {
 int  n, Sum, rev=0;

 printf("\n Enter the no. ");
 scanf(" %d", & n); a=n;
  while ( n!=0)
   {
 rev = rev* 10 + n % 10;
  n = n/10 ;
  }
   if ( rev= =a)
      printf("\n It is a palindrome");
  else
      printf(" \n It is not a palindrome");

      return(0);
      }
```

save a copy of n.

Program:- WAP to read a no. & find sum of its digits.

```
#include <stdio.h>
main()
{
int n, sum = 0;

printf("\n Enter the no.");
scanf(" %d", &n);
for( i=1; n/10 != 0; i++)
{Sum = sum+ n%10;
 n= n/10;
}
printf("\n Sum of digits = %d ", sum);
return(0);
}
```

Q. Write output.

```
i=0;
while ( i<5)
{
printf("%3d  %3d\n", i,10-i);
 i = i+1;
}
```

O/P:

```
0  10
1  9
2  8
3  7
4  6
5  5
```

Q. If n=10, find O/p.

```
scanf("%d", &n);
 ev = 0;
 while (ev<n)
  {
   printf(" %3d ", ev);
   ev = ev+2
   }
   printf("\n");
```

O/p:

```
0
2
4
6
8
```

★ Diff^t forms of a FOR loop

(i) Standard

(a)
```
for (i=1; i<=10; i++)
 {
  - - - -
 }.
```

(b)
```
i = 1;
for( ; i<=10; i++)
{
  - - -
}.
```

(c)
```
i = 1;
for( ; i<=10; )
{ i++;
}
```

(ii) Infinite for loop.

```
for( ; ; )
{ - - - -
}
```

(iii) Nested for

```
for(i=1; i<=10; i++)
{
    for(j=1; j<=2; j++)
    {
        printf("%d %d\n", i, j);
    }
}
```

o/p :-

```
1  1
1  2
2  1
2  2
 .
 .
 .
10 1
10 2
```

Program:- WAP to read a no. & find its factorian

```
#include<stdio.h>
main()
{
int i, n, prod = 1;
printf("\n Enter the no.\n");
scanf("%d", &n);
for(i=n; i!=0; i--)
{ prod = prod * i;
}
printf("\n The factorian is %d", prod);
return(0);
}
```

Program:- Enter a no. 'n' & find n^n.

```
#include< stdio.h>
main()
{
int i, n, prod = 1;
printf("\n Enter the no.\n");
scanf(" %d", & n);
for(i = 1 ; i <=n; i++)
 {
  prod = prod * n;
 }
  printf("\n The result is %d", prod);
return(0);
 }
```

SENTINAL
☆ ~~Setinal~~-controlled loops:-

eg: Find sum of all nos. input, till user inputs a negative number

```
# include< Stdio.h>
main()
{
int n=0, Sum=0, i;
while (n>=0)
  {
    printf("\n Enter the no. \n");
    scanf(" %d ", &n);
    sum = sum + n;
  }
    printf("\n Sum is %d" , sum);

}
```

* A FOR stmt. to implement a sentinel loop :-

```
for (scanf(" %d ", &num); num>0 ;
     scanf(" %d", & num))
   {
     s= s + num
   }
```

Program :- WAP to make the following asterisks pattern with n output lines. -

```
*
*  *
*  *  *
```

```
#include<stdio.h>
main()
{
int n, i, j;
printf("\n Enter the no. of lines");
scanf("%d", &n);
for (i=1; i<n; i++)
 {
     for (j=1; j<=i; j++)
      {
        printf(" * \t ");
      }
    printf("\n");
  }
}
```

* EXITING a loop with a BREAK STMT.

```
(i) while(...)
     { ....
         if (condn)
exit  ┌──── break;
      │    - - -
      │  } - ..
      └──→

(ii) do
      {
        - - -
        if (condn)
exit  ┌─ break;
      │    - - -
      │  }
      │  while (...);
      └──→

(iii) for(...)
       {
        - - -
        if (condn)
exit  ┌── break;
      │   - - -
      │  }
      └──→

(iv) for(- - -)
      {
        for(- - -)
         {
          ... if (condn)
       ┌──── break;
exit   │      - - -
from   │   }
inner  └──→ - - - -
loop  }
```

* EXITING a loop using GOTO STMT.

```
for (...)
  {
     for (...)
       {
         ....
         if (condn)
            goto aa;
            - - .
       }
  }
 aa:
   ....
```

exiting from 2 loops

* Skipping / Bypassing and continuing in loops.

* CONTINUE STMT:

(a)
```
while (test condn)
{
  - ..
  if (...)
   continue ;
  - - - -
}
```

(b)
```
do
{
  ---
  if (--)
   continue ;
  - - - -
}
while (test condn);
```

(c)
```
for (initializn; test condn; update)
{
  ---
  if (c...)
   continue ;
  - - -
}
```

Program :- WAP to print sq. root of n nos.

```
# include < stdio.h>
# include < math.h>
main ()
{
 int i, n, num;
 printf("\n Enter the no. of numbers \n");
 scanf(" %d", & n);

 for (i=1; i<=n; i++)
  {
   printf("\n Enter the number \n");
   scanf ("%d" & num);
     if (num < 0)
        continue;
  else
     printf(" The sq. root of number
     is %f \n", sqrt (num));
  }
}
```

Chapter - 3

Top down design with f^{ns} :

- C program supports modular programming also called Structural programming.
- F^{ns} : Library f^{ns} & user defined f^{ns}

↳ **Library functions**

· provides code reuse

- $\underbrace{\underbrace{y = \underset{f^n\ name}{sqrt}(x)}_{\to\ argument\ (or\ parameter)}}_{f^n\ call\ (f^n\ calling\ stmt.)}$

	F^n name	Header file	Argument type	Return Result t
1.	abs(x)	stdlib.h	int	int
2.	ceil(x)	math.h	double/ float	double/ float

ex :- ceil (45.23) = 46.0 (Round off to next value)

3.	cos(x)	math.h	double	double
4.	exp(x)	math.h	double	double
5.	fabs(x)	math.h	double	double

fabs (-8.432) = 8.432 → float abs.

6.	floor (x)	math.h	double	double

Opposite to ceil :- floor (45.23) = 45

7.	log(x)	math.h	double	double
8.	log 10(x)	"	"	"
9.	pow(x, y)	→ value of x^y		
10.	sin(x)			
11.	Sqrt (x)			
12.	tan(x)			

.... & others

User defined functions : 4 types.

1. f^{ns} without arguments
2. f^{ns} with arguments

} f^n can be called with/without giving any parameter.

3. f^{ns} without return value
4. f^{ns} with return value.

} the called f^n may/may not return a value.

1. Functions without arguments :

f^n call :

fname ( ) ;

f^n prototype / declar^n :

ftype fname (void);

→ type of f^n eg: void, int, float, etc.

f^n defin^n :

```
f^n header → ftype fname (void);
              {
                local variable declar^ns;
                executable stmts;
              }
```

Program: WAP which calls a f^n pow(x, y) power( ) to find x^y without arguments.

```
#include <stdio.h>
main()
{            ∵ not returning any value
                     ∵ no argument
  void power(void);     /* f^n declar^n */
  power() ;             /* f^n call */
```

* By default, all f^{ns} are assumed to return an integer value at their calling place. So, $declar^n$ not $reqd$ in that

```
/* f^n defin^n */
void power (void)
   {
      int x,y, v=1,i;
      printf("\n Enter values of x & y to
      find x^y.");
      scanf("%d %d", &x,&y);
      for(i=1; i<=y; i++)
        v = v*x;
      printf("The value of x^y is %d",v);
   }
```

2. Functions with input arguments

f^n call:

ACTUAL ARGUMENT

f^n name (list of input arguments);

f^n $declar^n$:

f type f^n name (data type of each actual argument);

f^n $defin^n$:

```
f type f^n name (type and name of each
                  formal argument) ;X
   {
      local declar^ns;
      executable stmt;
   }
```

Program: WAP to find x^y ~~f~~ using a f^n power( )

```
#include <stdio.h>
main ()
{
int x, y ;
void power (int x, int y) ;
printf ("\n Enter the values of x &y to
 find x^y \n") ;
 scanf (" %d %d", &x, &y) ;

 power (x, y) ;
 }
```

declare separately

```
 void power (int x1, int y1)
 {
    int x1, y1, i, prod = 1 ;
   for ( i=1, i<=y1 ; i++)
      {
      prod = prod * x1 ;
      }
     printf ("\n The value is %d", prod);
 }
```

The variables i & prod are local to this f^n called LOCAL VARIABLES.

4. Functions with a return value

```
#include <stdio.h>
main ()
{
 int  x, y, res ;
int pow (int, int) ;
printf ("\n Enter values of x &y ");
scanf("%d %d", &x, &y);

res = Power (x, y);
  printf("%d", res);
}
  int power (int x1, int y1)
{
   int v=1, i;
    for (i=1; i<=y1; i++)
        v=v*x1;
   return(v);
}
```

• Actual arguments	C	• Input arguments
• Arguments used to return a value	A	• Output arguments
• Default return value of a f^n	L	• int
• f^n returning no value	L	• void f^n
• f^n has to be declared at its calling place	E	• local f^n
• f^n declared outside main()	D	• ~~gl~~ global f^n.

Program: Using a f^n, 'swap()', WAP to exchange values to 2 variables a and b.
(with arguments)

```
# include<stdio.h>
main ()
 {
 int a, b;
void swap (int a, int b);
Printf("\nEnter the values of a & b to
swap them");
scanf("%d %d", &a, &b);
swap (a, b);
}

 void swap (int a1, int b1)
  {
   int a1, b1, temp;
   temp = a1;
    a1 = b1;
     b1 = temp;
  printf("\nThe swapped values of a
  and b are %d, %d", a1, b1);
  }
```

Program: Read 2 complex nos. & find their sum or difference or product depending on user's choice.
Use modular programming.

```
# include <stdio.h>
main ()
{
float rp1, rp2, ip1, ip2;
int  choice ;
void add(float, float, float, float);
void diff(float, float, float, float);
void prod( float, float, float, float);

printf("\n Enter the first number, i.e, its
real part and imaginary part");
scanf(" %f %f",&rp1,&ip1);
printf("\n Enter the second number, i.e, its
real part and imaginary part ");
scanf(" %f %f", &rp2,&ip2 ");
printf(" Enter the choice of operation to
be done 1. Addition, 2. Subtraction and
3. Multiplication");
scanf(" %d", &choice);
 switch (choice)
  {
     case 1 : add(rp1, ip1, rp2, ip2)
      break;
     Case 2 : diff (rp1, ip1, rp2, ip2)
      break;
     case 3 : prod (rp1, ip1, rp2, ip2)
       break;
  }
}
void add(float rp1, float ip1, float rp2,
float ip2)
 {
 float s1, s2;
  s1 = rp1 + rp2;
  s2 = ip1 + ip2;
 printf("\n The sum of the given 2
complex numbers is (%f)+i(%f)",
  s1, s2);
 }
```

```
void diff (float rp1, float ip1, float rp2,
float ip2)
{
 float d1, d2;
 d1 = rp1 - rp2;
 d2 = ip1 - ip2;
printf("\n The difference of given 2
complex numbers is (%f) + i(%f)",
d1, d2);
}
void prod(float rp1, float ip1, float rp2,
float ip2)
 {
  float p1, p2;
   p1 = (rp1 * rp2) - (ip1 * ip2);
   p2 = (rp1*ip2) + (ip1 * rp2);
printf("\n The product of 2 given complex
numbers is (%f) + i(%f)", p1, p2);
}
```

memory address

✡ Pass by value and Pass by reference

1. PASS BY VALUE :

```
main ()
{
 int a = 5, b = 10;
  void swap(int, int);
 swap (a, b);
 printf (" %d %d", a, b);
}
```

2(*(&a)
 *( )

```
swap(int a, int b)
{  int t;
   t = a;
   a = b;
   b = t;
   printf("%d %d", a b);
}
```

O/p :- 10 55 10

gives memory address

2. PASS BY REFERENCE : Using & and *

```
ex:- int a = 10;
     int *b;
     b = &a;
printf("value of a = %d", *b);
   o/p :- 10
```

→ way to declare a pointer variable

pointer variable ←

a → name

10 → value at the locⁿ

1000 → address

→ written before an address to access value of the variable 'b' within that address.

```
ex :- main()
      {
       int a=5, b=10;
       void swap(int *, int *);
       Swap(&a, &b);
       printf(" %d %d", a, b);
      }
     void swap( int *a1, int *b1)
      {
       int t;
       t = *a1;
      *a1 = *b1;
      *b1 = t; printf("%d %d", *a1, *b1); }
```

→ declarⁿ of a pointer

we are calling a fnⁿ by addresses of variables ←

→ value = &a

→ value = &b

(o/p:- 10 5)

Program :- WAP to read 3 nos. & sort them into ascending order, using a f^n.

```
#include <stdio.h>
 void order(float *small, float *large);
main()                                   Optional
 {
   float num1, num2, num3;
  printf("\n Enter the numbers : \n");
  scanf(" %f  %f  %f", &num1, &num2,
  &num3);
   order (&num1, &num2);
   order (&num1, &num3);
   order (&num2, &num3);
  printf("\n Numbers in ascending order are
  %f  %f  %f", num1, num2, num3);
 }
  void order(float *small, float* large)
  {
    double temp;
    if(*small > *large)
     {
        temp = *small;
       *small = *large;
       *large = temp;
     }
```

Trace of program :

Stmt :	num1	num2	num3
Scanf	7.5	9.6	5.5
order(& num1, & num2)		No change	
order(& num1, & num3)	5.5	9.6	7.5
order (& num2, & num3)	5.5	7.5	9.6

Program: WAP using a f^n to find n!.

```
# include <stdio.h>
main()
{
int fact(int n);
int n, v;
 printf("\n Read the value of n to
 find its factorial ");
 scanf(" %d", & n);
V=fact(n);
  printf("\n Factorial of n= %d", v);
}
int fact(int n1)
  {
     int f=1, i;
      for (i=1; i<=n1; i++)
          f = f*i;
       return(f);
  }
```

Program: ~~Red~~ Read n and r and find binomial coefficient $^nC_r \left(= \frac{n!}{r!(n-r)!}\right)$

```
# include <stdio.h>
main()
{
int n, r, num, den; float val;
printf("\n Enter the value of n and
 r to compute C(n, r) ");
scanf("\n %d   %d", &n, &r);
num = fact(n)
den = (fact(n-r))*(fact(r))
val = num/(float)den;

Printf("\n The value of C(n,r) is
  %f", val);
}
```

→ Same here.

i/p : n=2, r=1
o/p : 2.

Program: Find the trigonometric sine(x) given x and n

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - - - (-1)^{n-1}\frac{x^n}{n!}$$

Given:- x is in radians.

$180° = \pi^c$

$x° = (3.14/180) \times x$

```
# include <stdio.h>
# include <math.h>
main()
{
 int fact(int   );
 int n, sgn, i;
 float x, t, sin;
printf("\n Enter the values of x and
n for finding value of sin(x)\n");
scanf(" %f %d ", &x, &n);
* x = x*3.14/180;
 sin = x;
  sign = -1;
for (i=3; i<=n; i=i+2)
  {
        t = ((pow(x,i))/fact(i))*
                                  sign
  sin = sin + t
   sign = (-1)*sign;
  }
printf("\n Value of sin(X) is %f", sin)
}
```

define f^n fact here

Program: Read 3 nos. a, b and c and find their sum and avg. using a f^n calculate.

```
#include<stdio.h>
main()
{
                        declared
float a, b, c, sum, avg;
 void calculate (float, float, float, float*,
          float* & avg);
Printf("\n Enter the three values to
get their sum and average");
```

* value
& address.

```
scanf(" %d %d %d", &a, &b, &c);
calculate (a, b, c, &sum, &avg);
printf(" Sum = %f \n", sum);
printf("\nAverage = %f", avg);
}
 void calculate (float a1, float b1, float
  c1, float *s, float *av)
  {
     *s = a1 + b1 + c1 ;
     *av = *s / 3 ;
  }
```

f^n is called

→ garbage value gets stored in it.

Date ______
Page ______

Chapter - 8

ARRAYS

- derived data types
- collection of elements of same type.
- stores in conservative locations

ex:- marks [10]

array name → Index / subscript
integer constt / int var / int expn

→ marks [0] to marks [9] → 10 values

1D Array | 2D Array | Multi Dimensional

* 1D Arrays

- list of items - a single name
- elemts. accessed using just one subscript, so, its calle
- single subscripted variable.
- subscript begins with zero (0).

Declarⁿ:- datatype arrayname [size] ;
Keyword user defined +ve Z constt.

ex:- char name [10] : String
→ A character array.

* Storage :-

ex int a[3]

15	10	5
a[0]	a[1]	a[2]

values assigned
a[0] = 15
a[1] = 10
a[2] = 5.

- Array ~~ld~~ elements can be used like ordinary variables used in a C program.
  ex:- a[x] = a[0] + a[1] → subscript
  b = a[2] * a[2];
- Indexes of array (subscript) should be within declared limits.

Initializ^n:- datatype arrayname [size] = { list of values }

ex:- ◦ int number [3] = { 0, 0, 0 };

★ ◦ int number [3] = { 1 };

→ number [0] = 1
number [1] = 0 } automatically
number [2] = 0

~~Compile~~ Compile time Initializ^n

◦ int counter[ ] = {1, 1, 1}; → size would be taken as 3

size = 4+1 ← ◦ char name [ ] = { 'J', 'o', 'h', 'n', '\0' };

size = 4 + ① ◦ char name [ ] = " John ";

for null character

◦ char a [5] = { 'a', 'b', 'c' };

∴ a[0] = 'a', a[1] = 'b', a[2] = 'c'

★ Complete Initializ^n — int a[3] = {1, 2, 3}

Partial Initializ^n. — int a[3] = {1, 2}

★ String / character array

eg: char name[10]

value of name = " COMPUTERS ";

C	O	M	P	U	T	E	R	S	\0

'\0' : null character ; string terminator.

If char array is initialised later :

eg:

```
char name[5];            char name[5];
name = "JOHN";           strcpy (name, "JOHN");
      X                          ✓
```

- Char array is called a STRING.

* Runtime initializ^n

eg

```
int a[10];
for (i=0; i<=9; i++)
      scanf(" %d", & a[i]);
```

eg

```
float sum[100];
  for (i=0; i<=99; i++)
    {
      if (i<50)
           sum[i] = 0.0;
      else
           sum[i] = 1.0;
    }
```

elements from 0 to 49 are initialized 0.0. Resp. from 50 to 99 1.0.

Program: Read the marks ~~of 50~~ obtained by students of a class in a course & find course avg.

Static memory Alloc^n

```
#include<stdio.h>
main()
{                    → a max. size stored
int a[100]; int sum, n; char sub[15]; float
printf("\n Enter the no. of students in
the class \n");
scanf(" %d", & n);
printf("\n Enter the subject for which
marks would be entered\n");
scanf(" %s", sub);        → & not reqd for string
printf("\n Enter the marks \n");
for (i=0; i<=n-1; i++)
        scanf(" %d", &a[i]);
sum = 0;                          } can be
for (i=0; i<=n-1; i++)            } included in
        sum = sum + a[i];         } one for loop.
avg = sum/(float)n;
printf("\n The average marks of the class
in %s are %f", sub, avg);
}
```

Program:- Read marks of n students in a class & find highest marks in that course.

```
# include<stdio.h>
main()
{
int n, l;
int a [100];
printf("\n Enter the no. of students in the
class \n");
scanf(" %d", &n);
printf("\n Enter the marks \n");
     for (i=0; i<=n-1; i++)
          scanf(" %d", & a[i]);
avg;  l = a[0];  --> assume 1st value as largest.
     for (i=1; i<=n-1; i++)
        {
             if (l < a[i])
                  l = a[i];
        }
printf("\n The class highest is %d", l);
}
```

Program: Read a 1D array & reverse its elements.

```
#include<stdio.h>
main()
{
int a[50], n, t, i, j, k, l;
printf("\n Enter the no. of elements whose
order needs to be reversed \n");
scanf("%d", &n);
printf("\n Enter the elements ");
for (k=0 ; k<=n-1 ; k++)
    scanf("%d", & a[k]);
{for (i=0, j=n-1 ; i<n/2 ; i++, j--)
            t = a[i];
            a[i] = a[j];
             a[j] = t;
       }
  for (l=0 ; l<=n-1 ; l++)
     printf("\n The reversed elements are
  %d", a[l]);
}
```

Program: Read a string & check whether its a palindrome or not.

```
#include <stdio.h> #include <string.h>
main()
{
char name[20];
int i, j, n, flag=0
```

```
printf("\nEnter the string to check for
palindrome :\n");
scanf("%s", name);
n = strlen( name);
for ( i =0, j= n-1 ; i < n/2 ; i++, j--)
 {
  if (name[i] != name[j])
   {
      flag = 1;
      break;
   }
 }
if (flag == 0)
   printf("\nIts a palindrome\n");
else
   printf("\nIts not a palindrome\n");
```

Program: Read an array & count the no. of positive nos., no. of -ve nos. & zeroes.

```
#include<stdio.h>
main()
int n,num[50], i, j.

printf("\nEnter the number of elements of
 the array\n");
scanf("%d", &n);
printf("\nEnter the elements of the
 array \n");
```

```
for (i=0 ; i< n ; i++)
    scanf("%d",&num[i]);
for (j=0 ; j<n ; j++)
    {
        if
```

★ Call Arrays with functions

Calling an array : f^n name (array name, size).

f^n $defin^n$: f^n name (datatype array name [], datatype size)

Program:- Find sum of an array of n nos. using a f^n:-

```
#include <stdio.h>
main()
{
int a[5], n, i;               OR
★ void sum(int *, int);   void sum(int [], int);
printf("\n Enter the number of elements of
array \n");
scanf("%d", &n);
printf("\n Enter the elements");
for(i=0; i<=n-1; i++)
    scanf("%d", &a[i]);
sum(a,n);
}
void sum(int *b, int n)
{
                         OR void(int b[], int n)
int s=0, i;
  for(i=0; i<=n-1; i++)
  {
        s = s + *b;   OR *s=s+*(b+i);
        b++;
  }                   OR s += b[i];
   printf("%d", s);
}
```

☆ If array has 'n' nos.,
no. of iter^ns for sorting them are 'n-1'

Program :- Read a set of nos. & sort them in ascending order.

```
#include <stdio.h>
main()
{
int n, a[50], i;
int sort(int [], int);
printf("\n Enter the number of elements of array: \n");
scanf("%d", &n);
printf("\n Enter the elements of the array\n
for (i=0; i<=n-1; i++)
      scanf("%d", &a[i]);
# Print the elements before sorting in a for loop
sort(a, 50);
Print the elements after sorting in a for loop
}

int sort(int b[], int n)
{ int i, j, t;
    for (i=0, i<=n-2, i++)
    for (j=i+1, j<=n-1 ; j++)
      {
        If (b[i] > b[j])
          {
              t = b[i];
              b[i] = b[j];
              b[j] = t;
          }
      }
```

✫ 2D Arrays

a	col0	col1	col2	col3
row0	2	4	13	-12
row1	8	100	73	16
row2	68	52	49	15

a[0][0] = 2
a[1][2] = 73
a[2][3] = 15

- **Declaring a 2D array :**

Syntax: datatype arrayname [rowsize][col size];

ex:

①
```
int a[2][3];
char name [5][10]
```
array of char = STRING

②
```
# define R 2
# define C 3
main ()
{
    int a [R][C];
```

- **To access individual elemts. of 2D array :**

Syntax : arrayname [rowsubscript][col subscript]

ex:- a[0][1].

- **Reading a 2D array :**

ex:-
```
int a[2][3];
for (i=0; i<=1; i++)            <- For rows
{
    for (j=0; j<=2; j++)        <- For columns
    scanf("%d", &a[i][j]);
}
```
(Input of elemts.. is row-wise)

- **Initializing a 2D array :**

ex:-
```
int a[2][3] = {0, 0, 0, 1, 1, 1};
int a[2][3] = {{0,0,0}, {1,1,1}};
int a[2][3] = {
                    {0,0,0}
                    {1,1,1}
                };
int a[][3] = {
                    {0,0,0},
                    {1,1,1}
                };
int a[2][3] = {
                    {1,1}, {2}};
```

→ a[0][0] = 1 ; a[0][1] = 1, a[1][0] = 2

→ all of the other elements = 0 (automatically)

★ Multi Dimensional Arrays.

Syntax: type arrayname [S1][S2]....S[m];

ex - float table [5][4][3];

★ Static arrays: ex :- int a[5];

- created at compile time.
- size fixed. Cannot be modified.
- allocating memory at compile time → static memory alloc^n.

★ Dynamic arrays:

- vary in size.
- allocated memory at runtime.
- arrays created at runtime are called dynamic arrays.

- using pointer variables & memory management functions.— malloc(), calloc(), realloc() — <stdio.h>.

* Note:
when array is declared, all elemts are initialized to some garbage value.

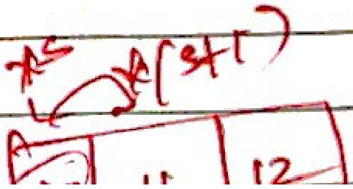
Program: WAP to read 2 matricies and find their sum

2

tally)

```
#include<stdio.h>
main()
 {
 int nra, nca, nrb, ncb, i, j, k, l;
 int a[][], b[][], c[][];
 printf("\n Enter the no. of rows & colm. of A\n");
 scanf(" %d %d", &nra, &nca);
 printf("\n Enter the rows & colms. of matrix B\n");
 scanf("%d %d", &nrb, &ncb);
 if (nra == nrb && nca == ncb)
     {
     printf("\nMatrix addition possible \n");
     printf("\n Enter elements of Matrix A\n");
     for (i=0; i<nra; i++)
         {
           for (j=0; j<nca; j++)
           scanf(" %d", &a[i][j]);
         }
printf("\n Enter the elements of Matrix B");
for (k=0 ; k<nrb ; k++)
    {
        for (l=0 ; l<ncb ; l++)
            Scanf("%d",&b[k][l]);
    }
for (i=0 ; i<nra ; i++)
    {
        for (j=0 ; j<nca ; j++)
            c[i][j]= a[i][j] + b[i][j];
    }
printf("\n The sum of matrices A &
    B is \n");
    for (i =0 ; i<nra ; i++)
      {
        for (j=0 ; j<nca ; j++)
          printf("%5d", c[i][j]);
          printf("\n");
      }
    }
else
    printf("\n Matrices cannot be
    added \n");
}
```

to give blank spaces b/w elemts. in a matrix



Chapter - 9

also called

CHARACTER ARRAYS & STRINGS.

- sequence of characters : string

⊙ Operⁿˢ on strings :

- Reading & writing strings
- Combining strings
- Copying one string to another
- Comparing strings for equality
- Extracting a portion of a string.

⊙ Declaration :

char stringname [size]

ex :- char city [10]

→ should include value of '\0'

'\0' : null character, automatically appended.

⊙ Initializⁿ :

ex: char city [9] = " NEW YORK" ;

char city [9] = { 'N', 'E', 'W', ' ', 'Y', 'O', 'R', 'K', '\0' };

char string [ ] = { 'G', 'O', 'O', 'D', '\0' } ;

* char str [3] = "GOOD"; } INVALID

* char str [5] ;
  str = "GOOD" ; } INVALID

* char str1 [4] = "abc" ;
  char str2 [4] ;
  str2 = str1 ; } INVALID

⊙ Reading strings :-

1) Using scanf()

ex:- char address[10];
scanf(" %s", address)
& not reqd

```
char address[10];
scanf(" %5s", address);
```

i/p : KRISHNA.

Store :- |K|R|I|S|H|\0| | | | |

unused locns :- assigned garbage va

- scanf() cannot read multiple words strings

To read all chars →

Syntax : char line[80];
scanf(" %[^\n]", line);
→ Reads a line of text till \n is input

2) Getchar() :-

- Reads just one char. So, use in while loop

ex:- char line[80]; char ch;

```
int c=0;
do
{
    ch = getchar();
    line[C] = ch;
    c++;
} while (ch != '\n')
```

format

3) Using gets() :- (Read multiword string)

ex :- char line [80];
gets (line);

- It accepts char. from i/p, assigns them to line, will not skip white spaces.

ex :- char line [10];

scanf ("%s", line);	gets (line);
Input :- abc efg.	
o/p :- abc	abc efg.

⑥ Writing strings :-

1. printf ("%s", name);
2. % wps → field width → actual posⁿ of string chars; no. of char to print
   ↳ (a) If w < string length : entire string is o/p
   (b) when p = 0 ⇒ no o/p.
   (c) %-wps → forced left justificⁿ.
   (d) %.ns : first n chars. of string.
   → field width not specified
3. char ch = " * " } char name [6]
   putchar (ch); } for (i = 0; i <= 4; i++)
   put char (name [i]);
4. puts (stringname);

☆ ARRAY OF STRINGS :-
2D array of char, in which each row is one string.

ex:- # define NUM-PEOPLE 30
# define NAME-LEN 25
- - -
- - .

30 rows   25 columns

char names [NUM-PEOPLE][NAME-LEN];

(each row is a string of size 25. i.e, 25 chars can be stored in each row)

ex:- char month [12][10] = {"Jan", "Feb", "Mar", "April", "May", "June", "July", "Aug", "Sept.", "Oct", "Nov", "Dec"};

* Arithmetic oper^{ns} on characters

1. char X;
   X = 'a';
   printf("%d\n", X);
   } o/p :- 97 (ASCII value of a)

2) X = 'z' - 1; ⇒ X = 122 - 1 = 121

3. if (ch >= 'A' && ch <= 'Z')
   - - - - - -
   (checks whether ch. is an uppercase alphabet)

4. ASCII values of A - 65   a - 97
   :   :
   Z - 90   z - 122

5) X = character – '0' ; (Converts a character digit to its equivalent integer value)
X = '7' – '0' ;
= 55 – 48
= 7

6. To convert a string of digits to its integer value :-

a to i (string)   stdlib.h.

ex :- char number [5] = "1998";
int year ;
year = atoi (number);
∴ year = 1998

☆ STRING HANDLING FUNCTIONS (string.h)

1. Strcat () : concatenates two strings

(a) strcat (string 1, string 2);

eg :- Part 1 :-

V	E	R	Y		\0					
0	1	2	3	4	5	6	7	8	9	10.

Part 2 :-

G	O	O	D	\0

strcat (part 1, part 2)

Part 1

V	E	R	Y		G	O	O	D	\0	

↳ Part 2 remains unchanged.
↳ Result comes in 1st string

(b) Str cat (string 1, "CONSTANT");
string constt

(c) strcat (strcat(string 1, string 2), string 3);
↳ nested strcat stmts.
↳ O/P in string 1.

2) strcmp() : compares 2 string arguments

strcmp(string 1, string 2)

↳ Value returned

↳ 0 (Strings equal)

↳ non zero (strings unequal)

-ve : String 1 is alphabetically above string 2.

returned value is diff. in ASCII values of the 2 chars compared (of strings).

eg :- strcmp(name 1, name); → +ve value returned

eg :- strcmp("ROM", "RAM"); ∵ ASCII$_O$ > ASCII$_A$

☆ if (name 1 == name)

↳ INVALID (can't compare strings like this)

3) strcpy() : Assign contents of string 2 to string 1

strcpy(string 1, string 2)

eg :- strcpy(city, "DELHI");

char city 1[6] = "DELHI";

char city2 [6];

4) strlen() :

n = strlen(string);

5) strncpy(S$_1$, S$_2$, n); → first n chars. of S$_2$ copied to S$_1$

eg :- strncpy(S$_1$, S$_2$, 5);

S$_1$[6] = '\0'; → First 5 chars. of S$_2$ copied to S$_1$

* Reqd to put null char.

6) strncmp() :-

strncmp(S$_1$, S$_2$, n);

Compares left most (first n) chars. of S$_1$ & S$_2$.

It returns : 0 : equal
-ve : S_1 substring < S_2
+ve : otherwise.

7) strncat( ) :

strncat (S_1, S_2, n);

Concatenates first n chars of S_2 to end of S_1

8) strstr( ) :

(11) A

strstr (S_1, S_2);

Searches S_2 in S_1

If S_2 found in S_1 : It returns pos^n of its first occurence.
else : null.

tring!

eg :- strstr(S1, "ABC");

strchr(S_1 , 'm'); → search for first occurence of char 'm' in S_1

strrchr(S_1 , 'm'); → Search for last occurence of char 'm' in S_1

→ search for a char in a string

search for a substring in a string.

• Strings : can be declared as pointers.
: can be passed as arguments.

★ Table of strings :

eg :- char name [10][5]
→ make an array of strings having 10 rows, each row has 5 chars.

★ substring : fragment of a longer string.

S1[0] S1[1] . . .

Jan.␢30,␢1996

```
eg :- char result[10], s1[5] = "Jan. 30, 1996";
   strncpy (result, s1, 9);
     result [9] = '\0';
o/p ;
   result :-      "Jan. 30, "

eg :- strncpy (result, & s1[5], 2);
        result [2] = '\0';
 o/p :-
   result :-      "30"
```

eg: To get final (few last) chars of source string.

```
       strcpy (result, & s1[9]);
ex :- # define STRSIZ 20
      char S1[STRSIZ] = "Jupiter ";
       char S2 [STRSIZ] = "Symphony";
  printf("%d %d\n", strlen(s1),
  strlen(strcat(s1, s2)) );
  printf ("%s\n", s1);

   o/p :-      8     16.
                 Jupiter Symphony
```

Program :- Read a string & check if its a palindrome.

```
#include<stdio.h>
# include< string.h>
main()
{
char word[30];
int  i, j;
```

Date 3.4.12
Page

```
printf("\n Enter the string to check for
palindrome \n");
scanf(" %s ", word);
n = strlen(word);
for (i=0, j=n-1; i<j; i++, j--)
    {
        if (word[i] != word[j])
            {
                printf("\n Not a palindrome \n");
                goto a1;
            }
    }
        printf("\n Its a palindrome \n");
    a1 : ;
}
```

or, use flag.

Program: Read a set of names and sort them into alphabetical order.

```
#include<stdio.h>
#include <string.h>
main()
{
char name[10][10], temp[10];
int i, j, n;
printf("\n Enter no. of names\n");
scanf(" %d", &n);
    for (i=0; i<=n-1; i++)
        scanf(" %s", name[i]);
for (i=0 ; i<=n-2 ; i++)
    {
      for (j=i+1 ;  j<=n-1 ; j++)
       {
        if (strcmp(name[i], name[j])>0)
         {
           strcpy(temp, name[i]);
           strcpy (name[i], name[j]);
           strcpy (name[j], temp);
         }
       }
    }.
for ( i=0; i<=n-1; i++)
       printf("\n The elements/names in
       ascending order are %s \n", name[i]);
}
```

☆ CHARACTER HANDLING FN

CHARACTER HANDLING FN^{S} :-

defined in : ctype.h.

char ch;

1. is alpha (ch) → checks whether ch is alphabet
   value returned = 0 or 1.

eg : if (isalpha (ch))
        printf(" ch is a letter");

2. isdigit (ch)
3. is upper (ch)
4. is lower (ch)
5. ispunct (ch) Checks whether argument is punctuation symbol
6. isspace (ch) : Checks whether ch is a space
7. toupper (ch) ;
8. to lower (ch) ;

Program: Read a line of text & count the number of ~~alphabt~~ alphabets, digits and words in it

```
#include<stdio.h>
#include <string.h>
#include<ctype.h>
main()
{                                                   count of
                                                    punctuation
char line[100];
int count1=0, count2=0, count3=0, i, n, cp;
printf("\n Enter the line of text \n");
scanf("%[^\n]", line);   or gets(line);
n= strlen(line);
    for(i=0; i<=n; i++)
    {  if(isalpha(line[i]))
             count1++;
       else if (isdigit(line[i]))
             count2++;
       else if (isspace(line[i]))
             count3++;
       else
             cp++;
    }
     count3+=1;  > no. of words would be 1 more than no. of spaces
printf("\n The no. of alphabets are %d \n The
no. of digits are %d \n The no. of words are %d
", count1, count2, count3);
}
```

ex:
```
char str[10] = "GOOD";
int i;
for (i=0; i< strlen(str); ++i)
    if islower(str[i])
        str[i] = toupper(str[i]);
```

✯ POINTERS

- a derived data type.
- have memory addresses as values.
- can be used to access and manipulate data stored in memory.

* Uses

- used to make a f^n return multiple values to its calling place through arguments (call by reference).
- makes f^{ns} to be passed as arguments to another f^n.
- used in dynamic memory management.
- ~~rob~~ reduces the length and complexity of programs.
- used in manipulating dynamic data structures- linked list, stacks, queues, trees..

* Pointer operators :

& : address of operator. &i

* : value of address *(&i)

≡ i

ex :-
```
main()
{
    int i=3.
    printf("\nAddress of i = %d", &i);
    printf("\n Value of i = %d", i);
    printf("\nValue of i = %d", *(&i));
```
o/p :-
```
Address of i = 6485
Value of i = 3
Value of i = 3
```

* Pointer variable

```
int i=3
j = &i; /*j is a pointer to variable i*/
```

* Declaring a pointer:

datatype *pointervariablename;

ex :- int *j; float *b; char *c;

* Valid / Invalid oper^{ns} :-

1.
```
float a,b;
int x, *P;
p = &a;  }
b = *P;  } INVALID
```

2. `int x, *p = &x;` VALID

3. int *p = &x, x;   INVALID

4) int * p = NULL   VALID
   int *p = 0;

5) int *p = 350;   INVALID

6) int x, y, z, *p;
   p = &x;
   - -
   p = &y;   VALID
   - - -
   p = &z;
   - - -

7) int x;
int *p1 = &x;   → multiple pointers
int * p2 = &x;   VALID   to same variable
int *p3 = &x;

ex. Change value of variable using pointer.

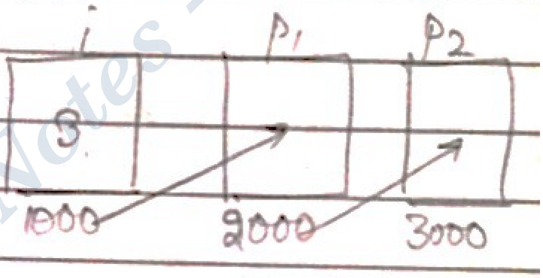
```
main ()
{
int x = 10, y;
int *ptr = &x;
 y = *ptr;
printf(" %d %d", x, &x);
printf(" %d %d", *&x, &x);
printf(" %d %d", ptr, *ptr);
printf(" %d %d", ptr, &ptr);
printf("%d %d", y, &y);
 *ptr = 25;
 printf("%d", x);
}
```

* Pointer to a pointer : CHAIN OF POINTERS.

a pointer points to another pointer ;

ex :-   → Called as Multiple indirection

```
int i = 3, *P1, **P2;
P1 = &i;
 P2 = &P1;
printf("%d", i);
prinf("%d", *P1);
printf("%d", **P2);
printf("%d", **(&P1));
```



* Pointer expressions :-

ex :- y = *P1 * *P2;   (*P1)*(*P2);

sum = sum + *P1;

Z = 5 * - *P2/*P1;

multiplicⁿ operator → unary minus

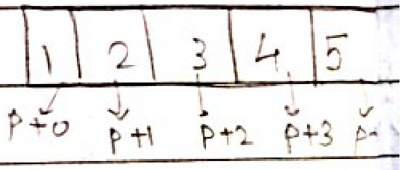
* Note : 2 pointers CANNOT be added, subtracted, multiplied, divided, ...

* pointers can be incremented. The incrementⁿ is based on their datatype (need not always be 1).

eg :- Create a pointer to an array.

```
#include <stdio.h>
main ()
{
int a[5] = {1,2,3,4,5};
int *p;
p = a;   /* Assign address of 0th element of
              a*/
for (i=1; i<=5; i++)
  {
    printf("%d\n", *p);
    p++;
  }   → p is incremented by 2 (∵ its an
           integer pointer)
}
```

p = &a[0] ≡



* SCALE FACTOR

When a pointer is incremented, it gets increased ~~on~~ depending on its type.

ex: /* make a f^n return a pointer to its calling place */

Chapter - 11

or record

STRUCTURES (Tuple Data)

- collection of dissimilar data types using a single name. (Its a derived data type)

ex: student : attributes of diff^t types

- name
- roll no.
- marks
- average
- grade

• Defining structures :

- define a structure.
- creation of structure variables.

key word

Format:
```
struct tagname  → user defined identifier
{
    datatype member1;
    datatype member2;
    . . . . .
};*
```

- a template is created to represent inform^n.

ex:
```
struct student
{
    char name[10];
    int age;
    float avg;
    char grade;
};
```

structure defin^n

struct student S_1, S_2 ;

→ structure declar^n for declaring structure variable

structure variable . member nam

→ member selection operator

eg. s1.name
S1. age

- Declaring structure variables

Format: struct tagname variable1, variable2, ...;

```
ex:- struct student
     {
       int total ;
       char  name[5] ;
       - - -
     };
  struct student  s1, s2;

ex(2):- struct student
        {
          int ID;
          char name [10];
        }s1, s2;

ex(3): struct
       {- -
        - -
       } s1, S2;
```

• no tagname.
• cannot be used for future declarations.

- Where to place structure definⁿˢ ?

• before all variable / fⁿ definⁿ.
• global definⁿ.

- Type defined structures

for renaming an existing datatype

Format : typedef struct

```
{
 . . . .
  type member1;
  type member2;
  - - - - -
} typename ;
```

just like name of structure

- To declare structure variables :

typename var1, var2, . . . ;

(in structure: struct structurename var1, var . . . . ;

- Accessing structure members :-

we don't need to use struct in typedefined structures.

Format :- structure variable name . member name

direct (member or component) selection operator

```
ex:- s1.id
     s1.name
ex:- strcpy (s1.name, "BASIC");
     scanf("%d", &s1.id);
     scanf ("%s", s1.name);
```

Program: Define a structure to store details of 2 students of a class.

- Initialising a structure
  At compile time.

ex :-
```
main()
  {
    struct record
     {
       int weight;
       float height;
     };
   struct record s1 = {60, 180.75};
   struct record s2 = {50, 160.5};
```

- Partial initializ^n is possible.

- Manipulating whole structures:
  copying and comparing structure variables:

structure copying

ex :- s1 = s2; — VALID

```
s1 == s2;  }
s1 != s2;  } INVALID
```

- No logical oper^{ns} on structure variables.
- Can be compared by comparing individual ~~members~~ members.
- Oper^{ns} can be done on members.

ex
```
if (s1.mark == 10)
        s1.mark += 10;
   sum = s1.mark + s2.mark
```

- We can apply increment/decrement operators on numeric type members.

ex Student.no ++;

- member operator (.) > rel^{nal} & logical : PRIORITY

– Arrays of Structures

ex.
```
struct marks
  {
    int maths ;
    int physics ;
  } ;
  struct marks students[50] ;
```

each element of this array is a structure.

- Initializ^n :-

ex
```
struct marks stu[3] =
  {{40,60}, {75,90},{19,20}} ;
```

stu[0].math	40
stu [0]. physics	60
stu [1]. math	75
stu [1] . physics	90
stu[2]. math	19
stu[2]. physics	20

– Structures within structure

- Nesting of structures : hierarchial structures

ex:
```
struct salary
  {
    char name;
    char dept ;
    int basic_pay;
    int da ;
    int hra ;
    int ca ;
  } employee ;
```

```
struct salary
{ char name;
  char dept;
  int basic_pay;
struct
  { int da ;
   int ca ;
    int hra ;
} allowance ;
} employee ;
```

To access innermost structure :
employee. allowance. da
employee. allowance. dra

ex

```
struct salary                           struct pay
{                                       {
char name[5];                           int da;
char dept [10];       declare outside   int dra;
int basic_pay;                          int ca ;
struct pay allowance;                   };
};
struct salary employee [50];
```

Nesting of more than one type of structures:

ex :-

```
struct salary
{                                       struct pay
char name[5];                           {- - -
char dept[10];                          };- -
struct pay allowance;                   struct d
struct d dob;                           {
};                                      };- -
struct salary emp[10];
```

Program: Define a structure to store ID no., name & 3 subject marks & total marks of each student in a class. Using this, WAP to prepare marksheet of all students in the class.

```
#include<stdio.h>
main()
{
int n, i;
struct student
{
int id;
char name[10];
int m1,m2,m3;
int total ;
};
struct student s[50];
printf("\nEnter no. of students \n");
scanf("%d", &n);
    for(i=0; i<=n-1 ;i++)
      {
        printf("\n Enter ID no.\n");
        scanf("%d", &s[i].id);
        printf("\n Enter name \n");
        scanf(" %s", &s[i].name);
        printf("\n Enter 3 subject marks");
        scanf("%d %d %d", &s[i].m1 , & s[i].m2
        , & s[i]. m3);
        s[i].total = s[i]. m1 + s[i].m2 + s[i].m3 ;
      }
printf("\nMarksheet \n\n\n");
```

①
↓

```
for (i=0; i<n; i++)
  {
    printf("\n Name \t : \t %s", S[i].name);
    printf("\n ID \t : \t %d", s[i].id);
    printf("\n Math \t : \t %d", s[i].m1);
    printf("\n Chem \t : \t %d", s[i].m2);
    printf("\n Phy \t : \t %d", s[i].m3);
    printf("\n Total \t : \t %d", s[i].total);
  }
}
```

Q. Given maruti.engine.bolts = 25;

(a) bolts $\xrightarrow{\text{member}}$ engine T

(b) engine → maruti T

(c) maruti → engine F

(d) maruti → bolts F

☆ – Passing an entire structure.

1. Each structure member can be passed as an actual argument of the f^n call.
   - inefficient when structure size is large.

2. Passing a copy of the entire structure to the called f^n.
   - all compilers don't support this.

3. Using pointers to pass structure as an argument.

ex ① Passing individual structure elements.

```
#include<stdio.h>
#include <string.h>
main()
 {
void display (char*, char*, int)
struct book
{
 char name[25];
 char auther [10];
 int call no;
 };
 struct book b1={"BASIC","YPK",101};
 display (b1.name, b1.auther, b1.callno);
 }
 void display (char*s, char*t, int n)
  {              → or: char s[10].
   printf("\n%s %s %d",*s,*t, n);
  }
```

ex ② :- Passing copy of entire structure.

structure declared outside main

```
struct book
 {
  char name[10];
  char auther[5];
  int call no;
 }
   main()
   { void display (struct book);
 struct book b1={"BASIC","YPK",101};
   display(b1);
 }
```

Passing entire structure ←

```
Void display (struct book b)
 {
   printf (" %s ", b. name);
   printf(" %s ", b. author);
   printf (" %d", b. callno);
 }
```

ex③ :- Passing structure with pointer

```
struct book
{
char name[10]
char author [5];
int callno ;
}
main()
{ struct book *p;
void display(struct book *);
Struct book b1 = {"BASIC", "YPK", 101};
p=&b1;
display(& b1);
          ↳ or p
}
void display (struct book *b)
{                                  → indirect method
printf(" %s" , b->name);             of accessing
printf(" %s" , b->author);           elements by
printf(" %d", b->callno);            pointers
}
```

* For a f^n returning a structure, datatype of f^n is struct structure-name f^n name (struct structure-name var name)

Q. Create a type defined structure to store following details of a planet :

Name
Diameter
Distance from moon
Rotation time

```
# include< stdio. h>
typedef struct
 {
   char name[10];
   double diameter;
   int moon-d ;
   double rot_time;
 } planet_t;
```

→ defined outside main()

new name given to struct

contd. Q. Using the structure datatype, declare a variable to store planet detail.

```
  main ()
  {
       * void print_planet (planet_t)
       planet_t current_planet;
       strcpy (current_planet.name, " JUPITER");
       current_planet.diameter = 142800;
       current_planet. moon-d = 10;
       current_planet. rot_time = 9.925;

       print_planet (current_planet);
  }
```

fⁿ declarⁿ

①

Name var name)

```
void print_planet(planet_t p2)
{
printf("Name = %s\n", p2.name);
printf("Diameter = %lf \n", p2.diameter);
printf("Distance from moon = %d\n", p2.moo);
printf("Time = %f \n", p2.rot_time
}
```

ex Call by REFERENCE

```
struct student
{
char name[10];
int id;
char grade;
};
typedef student s;
```

> rename structure student to s

or

```
typedef struct
{
char name[10]
int id;
char grade;
} s;
```

```
main()
{ void display(s);
  void scan_student(s*);
  s s1;
  scan_student(&s1); /* structure pointer */
  display(s1);
}
```

passing structure pointer to a fn

↓

```
));
void display (s  p2)
{ printf ("%s", p2.name);
  printf ("%d", p2.id);
  printf ("%c", p2.grade); }  → structure pointe
void scan_student (s * p2)
  {
    printf("\n Enter name \n");
    scanf(" %s", p2->name);
    printf("\n ID\n:");
    scanf("%d", P2 -> id);
    printf("\n Grade :\n");
    scanf(" %c", p2 -> grade);
```

Chapter -12.

FILE MANAGEMENT IN C.

- scanf() → to read data } When volume of
  printf() → to write data } data is small.
- when volume of data (i/p or o/p) is large, it is stored in disks — a secondary storage
- Concept of FILES is used to store data on disks
- File is a place on the disk, where a group of related data are stored.

* Oper^{ns} on FILES.

· naming a file
· opening a file
· reading from a file
· writing data to a file
· closing a file.

} done by using Std. library f^{ns} in C.

* Ways to do file oper^{ns}:

1. Low-level : uses UNIX sys. calls.
2. High-level : uses f^{ns} in C's std. ~~libr~~ library.

✲ Types of FILES

Text

- created using an editor or a word processor.
- named collection of chars stored in a disk
- no fixed size.
- eof (end of file): char to mark end of file. This is placed by the comp.
- all <enter> keys, when creating a file are replaced by \n chars.

ex This is a text file <newline> It has two lines. <newline><eof>.

Binary

It contains data in the form of 0 & 1.

✲ Defining and opening a file :

To store data in a file in secondary memory, we give following details about the file to the O.S :-

1. Filename
2. Data structure
3. Purpose.

1. FILENAME : String of chars, may have 2 parts — a primary name & extension (optional), separated by period.

ex :- Input.data
student.c
test.out .

2. DATA STRUCTURE : defined as a FILE in the std i/o f^n library.
∴ All files are declared as type FILE before used.

• General format for declaring & opening a file :-

3. PURPOSE

file pointer

```
FILE *fp;
fp = fopen("filename", "mode");
```

"r" -> read
"w" -> write
"a" -> append

f^n to open a file.
If file opens, f^n returns starting address of the file to the pointer fp.

* If fopen is failure, value of fp = NULL

Mode: r : open file for read only
w : " " " write only
a : " " " appending

* Each file should have its own pointer.
* Any no. of files can be used at a time.

Mode	File exists	File doesnt exist
w	• contents are deleted.	• a file with the specified name is created.
r	• file is opened with current contents safe.	• an error occurs. → Value of fp = NULL
a	• file is opened with current contents safe	• a file with specified name is created.

ex.
```
FILE *p1, *P2;
  P1 = fopen("data", "r");
  P2 = fopen("results", "w");
```

* Additional MODES:

r+ : existing file is opened for both reading and writing.

w+ : same as w, both reading and writing.

a+ : same as a, both reading and writing.

* CLOSING A FILE:

- when all oper^{ns} are completed, file is closed.
- all buffers - flushed out.
- all links are broken.

General format : fclose(file_pointer);

→ file pointer name (fp)

★ i/p & o/p oper^{ns} on FILES :-

1. The getc() & putc() f^{ns} :-

same as getchar() } used with std
putchar() } i/o.

• If ~~s~~ fp1 :- pointer to a file opened in write mode ("w");

c :- a character variable,

Then,

```
putc(c, fp1);
```

IIly,

getc() : reads character from a file opened in read mode.

ex

```
c = getc(fp2);
```

★ File pointer is moved by one character pos^{n}, after every getc() & putc()

getc() : returns ~~eof~~ EOF, when end of file is reached.

EOF : last char written to a file.
Press Ctrl+Z to mark end of file.

Program: Create a text file.
Write the contents of the file created onto the standard o/p device (monitor).

* EOF ⟺ ^z

Ctrl+Z

```
#include<stdio.h>
main()
 {
  FILE *fp;
  char ch;
  fp = fopen(" sample.txt ", "w");
   if (fp ==NULL)
     {
      printf("\nError");
       exit(0);
     }
     else
    ch= getchar();
   while (ch != EOF)  } or while ((ch=getchar())
     {                                   != EOF)
      putc(ch, fp);     |      putc(ch, fp);
       ch=getchar();    |
     }                  }
                        /* File now created */
 /* Now, open file in read mode, close first */
   fclose(fp);
    fp = fopen(" sample.txt", "r");
  ch = getc(fp);
  while(c !=EOF)
    {
       printf(" %c \n", c);
        c = getc(fp);
    }
       fclose(fp);
 }
```

✲ getw() & putw() functions.

- used only with integer data.

General format
putw (integer, fp); } used only for file
getw (fp); } with integer contents

ex:- Read 10 integers and write them into a file.

```
#include <stdio.h>
main()
{
 FILE *f1;
 int num, i;
 f1 = fopen(" Num.dat", "w");
   for (i=1; i<=10; i++)
     {
      printf("\n Enter a no. \n");
      scanf(" %d", &num);
      putw(num, f1);
     }
 fclose (f1);
/* Now, read contents of file & display o/p */
f1=fopen (" Num.dat", "r");
   for (i=1; i<=10; i++)
      {
       num= getw(f1);
       printf(" \n %d", num);
      }
  fclose(f1);
}
```

ogram: WAP to store 10 integers into a file NUM.DAT. Read each integer from NUM.DAT and store only odd integers in a file ODD.DAT & even integers in a file EVEN.DAT.

```
#include<stdio.h>
main()
 {
  FILE *f1, *f2,*f3;
  int num, i;
  f1 = fopen("NUM.DAT", "w");
      for (i=1; i<=10 ; i++)
         {
          printf("\n Enter no. \n");
          scanf(" %d", &num);
          putw(num, f1);
         }
      fclose(f1);
  f1 = fopen("NUM.DAT", "r");
  f2 = fopen("EVEN.DAT", "w");
  f3 = fopen("ODD.DAT", "w");
  for (i=1; i<=10; i++)
  {
  num = getw( f1)
      If ( num % 2 ==0)
         putw(num, f2);
      else
         putw(num, f3);
     }
 fclose(f1);
 fclose(f2);
 fclose(f3);
```

/* To print the file contents */

```
ex. f2 = fopen("EVEN.DAT", "r");
      while ((num = getw(f2)) != EOF)
          printf("%d\n", num);
```

write a record to a file.

* fprintf() and fscanf() :-
can handle a group of mixed data simultaneously.

General format : fprintf(fp, "controlstring", list);

to enter grp of data of diff' types simultaneously

→ file pointer of the file which is opened for writing.
→ o/p specific^ns of items in the list

constt, var, & strings.

ex. fprintf(f1, "%s %d %f", name, age, 7.5);
; name : array of char, age : int variable.

General format fscanf(fp, "control string", list);

to read a record ex. fscanf(f2, "%s %d", item, &qty);

Program: Create a file named STUDENT.DAT to store details of each student (5) in a class. Use this file to calculate & print total marks. Store: ID, name, m1, m2.

```
#include <stdio.h>
main()
{
 FILE *fp;
 float tot ;
 struct stu
{
 char name [10];
 char idno [10];
 float m1, m2 ;
 };
 struct stu s[50] ; /* Array of structures */
/* Open file */ fp = fopen(" STUDENT.DAT", "w");
  /* Now, make user enter elements */
  for (i=0; i<=4; i++)
 {
   printf("\n Enter ID no.\n");
   scanf(" %s" , s[i]. idno);
   printf("\n Enter name \n");
   scanf(" %s", s[i]. name);
   printf(" Enter mark1");
   scanf(" %f", & s[i]. m1);
   printf("\n Enter mark 2");
   scanf(" %f", & s[i].m2);

fprintf(fp," %s \t %s \t %f \t %f \n",
 s[i].idno, s[i].name, s[i].m1, s[i].m2);
 }
    /* File has been created */
```

①
↓

```
fclose (fp);
/* To calculate total & print */ /* Marksheet */
  fp = fopen (" STUDENT.DAT ", "r");
    for (i=0; i<=4 ; i++);
     {
  fscanf(fp, " %s %s %f %f", s[i].idno,
     s[i].name, &s[i].m1, &s[i].m2);
tot = s[i].m1 + s[i].m2;
 printf ("Name : %s \t", s[i].name);
printf ("ID No : %s \t", s[i].idno);
printf("\nMark1 : %f\t", s[i].m1);
printf ("\nMark2 : %f \t", s[i].m2);
printf("\nTotal : %f\t", tot);
     }
 fclose (fp);
}
```

Program: Assume "pr1.c" is an existing text file. Copy its contents character by character to another text file called "pr2.c".

```
# include <stdio.h>
main ()
{
 FILE *fp1, *fp2; char ch;
fp1 = fopen ("pr1.c", "r");
 if (fp1 == NULL)
    {
     puts ("\nCannot open source file \n");
     exit ();
```

```
fp2 = fopen("pr2.c", "w");
if (fp2 = = NULL)
{
  puts("\n Cannot open target file \n");
  exit();
}
fclose(fp1);
      do
      {
      ch= getc(fp1);
      putc((ch, fp2);
      }
      while (ch != eof);
fclose(fp1);
fclose(fp2);
}
```

or

```
while (1)   a true
{
ch= getc(fp1);
if (ch= = EOF)
  break;
else
  putc(ch, fp2);
}.
```

each file should be closed separately

→ o/p → nothing. Contents of files have to be seen by opening those files.

Program :- WAP to ~~read~~ store every character input from the keyboard into a file. The procedure should come to an end when the character "~" is entered from keyboard.

```
#include
main()
{  FILE* fp;
   char ch;
  fp = fopen("temp.dat", "w");
while (1)
 {
  getchar(ch)
  if (ch= = '~')
        break;
```

rintf (fp1;"

~~fsca (fp1,~~

FILE

= fop

```
else
    putc(ch, fp);
}
   putc(ch, fp);  → for printing the last
                    char '~'.
fclose(fp);
}
```

Program: WAP to count the no. of words in a text file - "sample.text".

```
#include<stdio.h>
main()
{
 FILE *fp;
 char ch; int count = 0;
fp = fopen("sample.text", "r");
 while(1)
  {
    ch = getc(fp);
    if (ch == ' ')      } or  if (isspace(ch))
       count += 1.      }       count += 1;

    if (ch == EOF)
        break;
  }
```

——→——

* Fns that access stdio and ~~st~~ stdin.
  - scanf(), - printf() - getchar(),
  - putchar().

* Fns that access any text file :
fscanf() , fprintf(), getc(), putc()

ex Using a string in fopen stmt.

```
char filename[50];
scanf(" %s", filename);       } where
fP = fopen(filename, "r");    } i/p:Dr1.c.
```

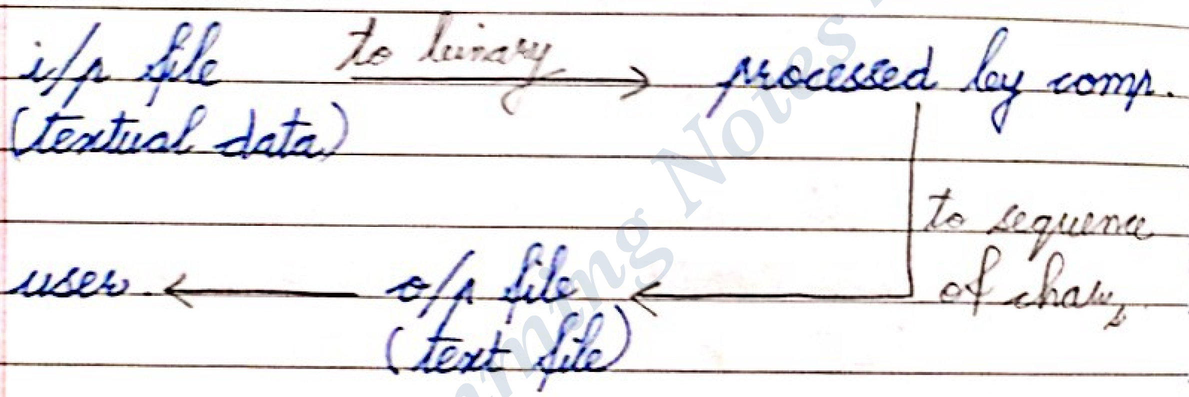
Program :- Make backup copy of a text file.

Cond'nal operator used in for stmt.

```
# include< stdio.h>
main()
{
  char in_name[80];
  char out_name[80];
  FILE *inp, *outp;

  printf(" Enter filename to be backed \n");
   for ( scanf("%s", in_name);
    (inp = fopen( in_name, "r")) == NULL;
    scanf("%s", in_name))
   {
     printf(" Cannot open file \n");
     printf(" \n Re-enter file name ");
   }
  /* get filename to store backup */
  printf(" Enter name for backup copy ");
  for (scanf("%s", out_name);(outp = fopen
  (out_name, "w"))= = NULL; scanf("%s", out_name)
   { printf(" Re-enter file name \n");
   }
/* copy chars (one at a time) */
for (ch = getc(inp); ch != EOF; ch = getc(inp))
    putc(ch, outp);
    fclose(inp);
    fclose(outp);
}
```

* Binary File :



i/p file (textual data) → to binary → processed by comp. → to sequence of chars → o/p file (text file) → user

- If o/p file of program 1 is i/p file to program 2, the transl^ne can be avoided, by using binary files, instead of text files.

- Binary file :
  file having binary nos. that are computer's internal represent^n of each file component.

ex.
```
FILE * binaryp ;
int i ;
binaryp = fopen("nums.bin", "wb");
    for (i = 2 ; i <= 500 ; i += 2)
        fwrite (&i, sizeof(int), 1, binaryp);
           → address  → size of        → no. of values
                        1 such values
    fclose (binaryp);
```

* 1 fwrite stmt. can be used to store/write array.

ex:- To write array of 10 Z_s to a binary file
fwrite(score, sizeof(int), 10, binaryp).
↳ array name

* fread, fwrite : used to read and write into a binary file

* fwrite : 4 arguments

① Address of (1st) memory cell whose contents would be copied to the file.
(& variablename, array name, & structure)
array of structures

② Size of 1 value to be written to the file

int	2 bytes
float	4 bytes
char	1 bytes

③ No. of values to be written.

④ Pointer to the file being created.

IIly for fread()
↳ returns how many values it has successfully read from the file

Disadvantages of Arrays.
ME

☆ Dynamic Memory Allocation

- Data : dynamic in nature.
- no. of data items may change during program execution.

ex : list of student names.

- Use dynamic data structures and dynamic memory management.
- Dynamic data structures : to add, delete or rearrange data items at runtime.
- Dynamic memory management : allocates add^{nal} memory space or release unwanted memory space at runtime.

Disadvantages of Arrays.

- Memory allocated at compile time :- STATIC MEMORY ALLOC^n.
- → int a[10]; → specify array size at compile time
- → wastage of space.

☆ LIBRARY FUNCTIONS

Memory Alloc^n f^{ns}:

F^n name	Task
1. malloc returns a void pointer	allocates requested size of bytes and returns a pointer to the first byte.
2. calloc	allocates memory for an array of elements, initializes them to zero returns a pointer to zero it.

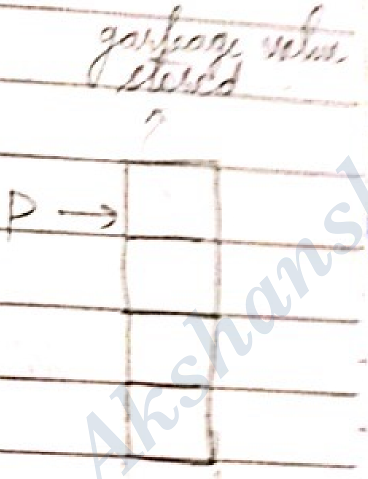
1.* malloc : returns a void pointer. So, we have to typecast it accordingly.

ex.

```
int *p ;
scanf("%d", &n);
p = (int *) malloc (n*2);
```

typecasting done

memory to be reserved for n integers.



* Fn name	Task
3. free	frees previously allocated space
4. realloc	modifies the size of previously allocated space.

* HEADER FILE : < alloc.h>.

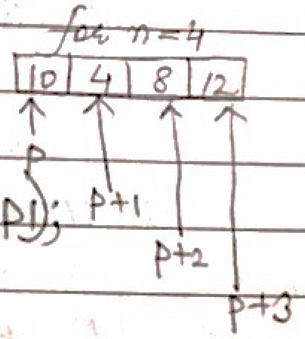
ex Read an array of n elements.
(i) find their sum
ii) Print elements in reverse order.

```
#include <stdio.h>
main()
{
int n, *p, sum=0, *p1;
printf("\n Enter size of array \n");
scanf("%d", &n);
                                        > or 2.
p = (int *) malloc(n* sizeof(int));
if (p==NULL)
  {
   printf("Insufficient space"); /*for check*/
   exit();
  }
printf("Add of 1st byte %u", p);
printf("Read array elements \n");
   for (p1=p; p1<=(p+n-1); p1++)
      {
        scanf("%d", p1);
        sum= sum + *p1;
      }
/* For reversing : */
for (p1=p+n-1; p1>=p; p1--)
  {
   printf("%d at address %u", *p1, p1);
  }
printf("%d", sum);
}
```

for unsigned Z

from 0 to n-1 : p1 points to addresses of p1

for n=4



O/P:

```
   Enter size of array   5
   Address of 1st byte   2252
   Read array elements
   1  2  3  4  5.
5 at address   2260  ← p+4
4 at address   2258  ← p+3
3 at address   2256  ← p+2
2 at address   2254  ← p+1
1 at address   2252  ← p
```

Sum, print at last

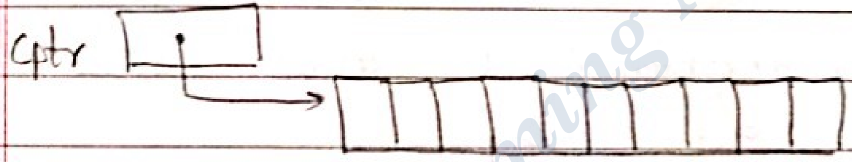
* When it is not possible to allocate enough space malloc() returns NULL.

* Allocating a block of memory : malloc()
- malloc() returns a pointer of type void.

General format : ptr = (cast_type *) malloc(byte_size);

ex :, X = (int *) malloc (100 * sizeof(int));
allocates 100x2 = 200 bytes of memory & returns a pointer to first byte.

ex: cptr = (char *) malloc(10);



10 bytes of memory.

ex : st_var = (struct store *) malloc(sizeof(struct store));
reqd to find size of Structure datatype

2. calloc()
- Allocates multiple blocks of memory. Blocks are of same memory.
- ~~Storims~~ Storing derived data types like arrays and structures.
- All blocks allocated are set to zero

General format : ptr = (cast_type *) calloc(n, elemt_size);
n : no. of elemts/blocks.
elemt_size : size of each block.

• Unsuccessful : returns NULL.

eg. store an array

int a[10];

p=(int *) malloc(10*2);   p=(int*) calloc(1, 10*2)

3. realloc():

- altering the size of a block.
- If original alloc^n is done by the stmt -

ptr = malloc(size);

realloc^n can be done by

General format: ptr = realloc(ptr, newsize);

starting address of previously allocated memory

- no guarantee that realloc^n takes place in same old region.
- If no addn^nal space, in the same old region, an entirely new ~~re~~ region of size newsize is created and contents of old block are moved to the new block.
- old data is safe.

Program: Using malloc() f^n, store the string "HYDERABAD" & then, modify the same memory block to store a larger string "SECUNDERABAD"

```
# include< stdio.h>
# include< alloc.h>
main()
{
```

```
char *buffer;
  if ((buffer = (char *) malloc (10))= = NULL)
   {
     printf("\nMalloc() failed");
     exit();
   }
strcpy (buffer, "HYDERABAD");
                  starting address of allocated memory
printf("Buffer has %S \n", buffer);
  /* Realloc^n */
if((buffer = (char *) realloc(buffer, 13))= = NULL)
   {
     printf("\nRealloc^n failed");
     exit();
   }
printf("Buffer has %s \n", buffer);
strcpy(buffer, "SECUNDERABAD");
printf("Buffer now has %S \n", buffer);
free (buffer); → returns free memory of buffer
}
```

original string was moved. So we are checking that.

Program: WAP to store the array int a[5] = {1,2,3,4,5} using dynamic memory alloc^n f^ns. Then, extend memory to store 2 more integers, 6 and 7.

```
#include<stdio.h>
#include<alloc.h>
main()
 {
```

```
int *p, *pl , i=1;
if ( p=(int *)malloc(5*2)==NULL);
   {
   printf("\nMalloc failed \n");
   exit();
   }
 for ( pl=p ; pl<=p+4 ; pl++)
   {
      *pl=i;
       i++;
   }
```

For checking

```
 printf("\n Before allocation \n");
 for (pl=p ; pl<=p+4; pl++)
     printf("%d", *pl);
```

let 5 values transferred already

```
 p=(int *) realloc(p, 7*sizeof(int));
   i=6;
 for(pl=p+5 , pl<=p+6 ; pl++)
   {
     *pl = i ;
      i++ ;
   }
 printf("\n After realloc" \n");
  for (pl=p ; pl<=p+6 , pl++)
     printf(" %d", *pl);
}
```

Program:- Using dynamic memory alloc^{n} f^{ns}, WAP to store the string BITS. & extend the memory to store the string BITS PILANI.

```
# include < stdio.h>.
main()
{
char * buffer;
if ((buffer = (char *)malloc(4)) = = NULL)
   {
    printf("\n Malloc failed\n");
    exit();
   }
strcpy(buffer, "BITS");
printf(" Buffer has %s \n", buffer);
   if ((buffer=(char *) realloc(buffer, 11))=
   = NULL)
   {
   printf(" \n Reallocation failed \n");
   exit();
   }
printf("\n Buffer has %s \n", buffer);
strcpy( buffer, " BITS, PILANI" );
printf(" \n Buffer now has %s \n", buffer);
     free(buffer);
}.
```

☆ Stacks

ex. Top → 5 ← most recently received item → gives index of topmost element
4
3
2
1 ← earliest item received.

* LIFO : Last In First Out.

Defin^n: Ordered list of zero/more elements which allows insertions and deletions at only one end called the TOP.

Implement^n: Using an array
or Using a LL (link list).

* Array implement^n of a stack.
Empty stack : top = 0
Full stack : top = array size

* Oper^ns done on a Stack:
- PUSH : adding an elemt. to top end of stack.
- POP : removing an item from top end of stack.

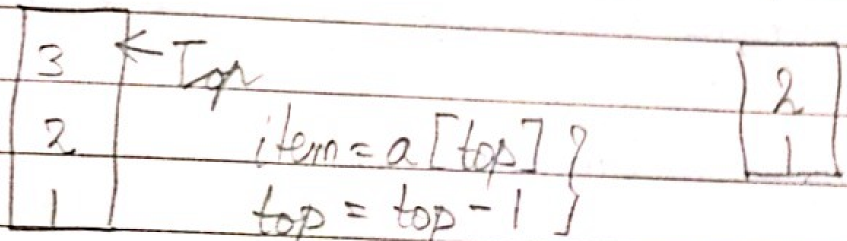
- Using arrays:

Before push.
3 ← TOP
2
1

Push item 4
top = top + 1
a[top] = item

After push
4
3
2
1

Before Pop After Pop.



item = a[top] ;
top = top - 1]

item popped is 3

* Algorithm to push an item onto the stack:

```
1. Read item to be pushed.
2. Check for stack overflow.
   If stackpointer, top, is greater than array-
   -size, ∃ stack overflow.
   So, print stackoverflow and stop else
   goto next step.
3.   top = top + 1
   a[top] = item .
4. Stop .
```

* Algorithm to pop an item from top end of the stack.

```
1. Check for empty stack (top = 0).
   If top is zero, print stack is empty and
   stop . Else goto next step
2. Pop the item on the top end of stack .
   item = a[top]
   top = top - 1
3. Stop .
```

* Applications
handling recursions, fⁿ calls & compiling.

* Implementation in C.

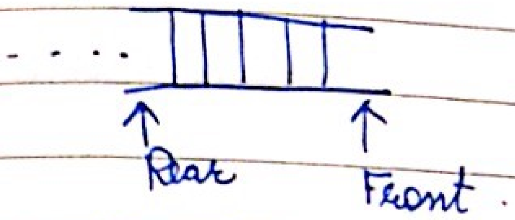
```
#include<stdio.h>
main()
{
int a[10], top, item, ch;
* top = -1;  ---> For finding empty stack (∵ array starts from 0
printf("\n1...>Creation\n");
printf("\n2...>Push\n");
printf("\n3...>Pop\n");
printf("\n4...>Display\n");
printf("\nEnter your Choice\n");
scanf("%d",&ch);
    while(ch<5)
    {
                                  a fⁿ
    switch(ch)
     {
      case 1:  top = create(a,top);
                   break;
      case 2 : top = push(a,top);
                 break;    > fⁿ
      Case 3 : top = pop(a,top);
               break;
      Case 4: display(a,top);
     }
printf("\n Enter user choice \n");
scanf(" %d", &ch);
}
} /* end of main */ /* Now f^n defin^ns */
   int create (int a1[10], int t)
      {
        int item;
   printf("\n Enter the elements (type 0 to
   terminate) \n");
   scanf (" %d", & item);
   while (item != 0)
   {
       t ++;
★      a1[t] = item;
       scanf(" %d", &item);
     }
   return (t);
} int push (int a1[10], int t)
{ int item ;
printf ("\n Enter element to be pushed");
scanf(" %d", & item);
  if (t>9)
        printf("\n Stack overflow \n");
  else
     { t++;
        a1[t] = item;
     }
       return(t);
  }
```

```
int pop (int a[10], int t)
{
  int item ;
    if (t>=0)
    {
      item = a[t] ;
       t -- ;
    printf("\n Popped item is %d\n",item)
    }
   else
    printf("\n Stack is empty \n");
    return (t);
  }
void display (int a[10], int t)
  {
    int i;
      for (i=t; i>=0; i--)
        {
          printf("\n\t%d\n", a[i]);
          printf("......\n");
        }
  }
```

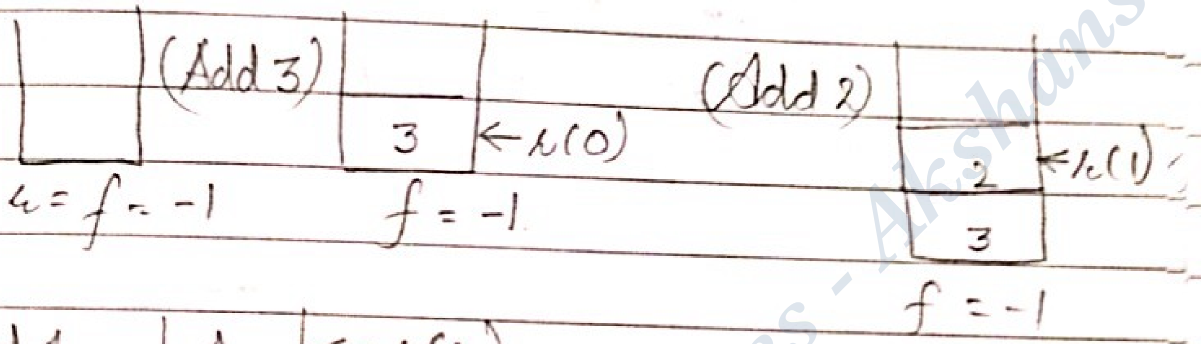
☆ QUEUE

- FIFO: first in first out.
- has 2 pointers, rear and front.
- Insertions take place at rear end.
- Deletions take place at front end.

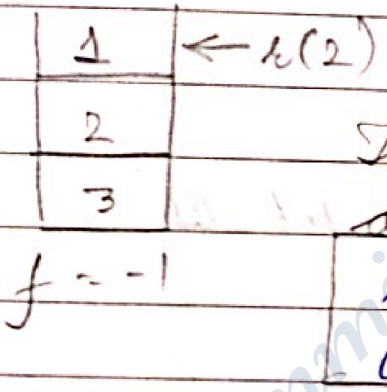
Array implementation of a queue :



Initially, front = rear = -1.



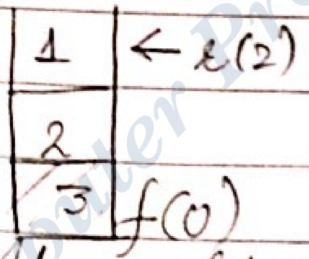
Add 1.



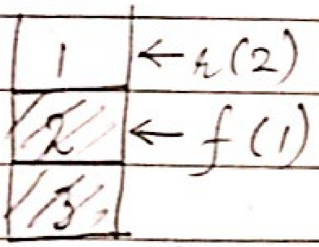
To add a data item to a

rear = rear + 1
a[rear] = item

* Delete Q



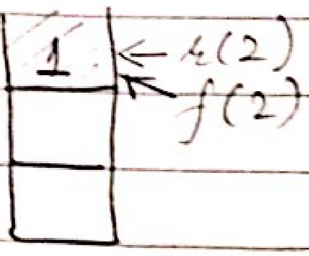
Del Q



Item deleted 3.

Item deleted = 2.

Del Q.



f = f + 1
item = a[f]

* Empty Queue — Full Queue

front == rear — rear = arraysize → some integer

eg

```
#include <stdio.h>
int front, rear;  → global variables
main()
{
int a[10];
void addq(int[], int);
int delq(int[]);
void qdisplay(int[]);
int i, j, d, ch;
front = rear = -1 /* Initialize */
system("clear"); /* Clrscr*/
while (ch != 4)
 {
 printf(" 1 --> Add ");
 printf("\n2 --> Delete \n");
 printf("\n3 --> Display \n");
 printf("\n4 --> Quit \n");
 printf("\n Enter your choice \n");
 scanf("%d", &ch);
 switch (ch)
 {
  case 1 :
            printf("Enter value to insert \n");
```

```
          scanf("%d", &d);
          addq(a, d);
          break;        → Add d to array a
    case 2:
          printf("\n Deleted item is %d",
          d= delq(a));
          break;
    case 3:
          qdisplay(a);   /* Call fn qdisplay */
          break;
case 4:
exit();
       }  /* switch close */
     }  /* while loop close */
   }  /* main fn close */
  /* Now, fns to be defined */
  void addq(int a1[10], int d1)
       {
          int r;
          if (r==9)
               printf("Full Q");    /* Check for full Q */
          else
               {
               r++;
               a1[r] = d1;      or   a1[++r]=d1;
               }
       }
```

fn is returning the deleted value at its calling place

```
  int delq(int a1[10])
     {  if(r==f)  /*Check for empty Q */
int r,f;   printf("Empty Q");
        else
             return(a1[++f]);
     }
```

```
void qdisplay(int al[10])
  {
     int i;
     if (front==rear) /* Check for empty */
     printf("Empty Queue.");
     else
       {
         for(i=front+1; i<=rear; i++)
            printf("%d\t", al[i]);
       }
  }
```

Set a counter & increment it (if size/no. of items in a queue needs to be known).

* Arguments to function main()

Two arguments to main():-

int argc ..→ an integer.

char *argv[] ..→ an array of pointers to strings.

--- command line arguments.

Program: File backup using arguments to main():-

```
/* make backup of the file whose name is the
first command line argument. Name of the
new file is second command line argument */
#include <stdio.h>
#include <stdlib.h>
int main( int argc, char *argv[])
  {
```

not necessary (optional)

no. of parameters stored in cmd line

```
FILE *inp, *outp;
                                      > source file
char ch;
inp = fopen(argv[1], "r");
if (inp == NULL)
    { printf("\n Cannot open the file \n");
      exit();
    }
                                      > Destin" file
 outp = fopen(argv[2], "w");
 if (outp == NULL)
     { printf("\n Cannot open file \n");
       exit();
     }
  for (ch=getc(inp); ch!=EOF; ch=getc(inp))
        putc(ch, outp);
  fclose (inp);
  fclose (outp);
}
/* To compile :- $ cc backup.c
   To run :- $ a.out ex1.c ex2.c
     argc = 3 <-
     argv[1] = ex1.c
     argv[2] = ex2.c                    */
```

Program: Find the sum of all nos. passed as arguments to the program from command line

```
#include<stdio.h>
#include<stdlib.h>
main (int argc, char * argv[])
```

```
{
int sum = 0, i;
  if (argc == 1)
    {
      printf("usage: %s [no1][no2] ...\n",
       argv[0]);
      exit(0);
    }
printf("argc = %d \n", argc);
  for (i=0; i<argc; i++)
      printf("argv[%d]= %s \n", i, argv[i]);
sum = 0;
  for (i=1; i<argc, i++)
      sum = sum + atoi (argv[i]);
printf("sum = %d \n", sum);
}
```

to print each argument in command line.

Run time: | $ cc sum.c
          | $ a.out 1 2 3 } or

```
sum      10   11   12   13
argc = 5
argv[0]=sum     argv[3]=12
argv[1]=10      arg[4]=13
argv[2]=11
$ cc sum.c -o sum
$ ✓sum 1 2 3
```

use the filename directly, instead of a.out argv[0] argv[1] argv[2]

i.e., a.out abcdefabc123 b → will be declared as a string

o/p :- aBcdefaBc123

Program: WAP to input a string & a character as command line parameters & convert the occurences of the character in the string to its upper case.

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h> /* check this {?} */
main (int argc, char *argv [])
 {
   char str[15], ch; int n, i;
   if (argc != 3)
    { printf("\n Error in i/p");
      system ("exit");      → or exit(0);
    }
   strcpy(str, argv[1]);
   ch = argv[2][0];   → 0th char. stored in argv[2]
   n = strlen(str);
    for (i=0; i<n; i++)
     {
        if (str[i]==ch)
         str[i] = toupper(str[i]);
     }
   printf("%s", str);
 }
```

Program: WAP which takes 2 nos. & an operator from the cmd line & gives the result of applying the operator on the 2 nos.

eg: $a.out 10 12 + → argv[3] = "+".

→ everything stored as a string

o/p: 22

```
#include<stdio.h>
# include<stdlib.h>
#include <alloc.h>      > for atoi , toupper, - - -
main(int argc, char *argv[])
  {
 char op, str[15], ch; int i, a, c
     if(argc != 4)
        { printf("Error in i/p \n");
          system("exit ");
        }
   a= atoi(argv[1]);   > string => requires
                         typeconversion
   b = atoi(argv[2]);
   op= argv[3][0];
  Switch(op )
      {
         case '+' :
            printf(" %d", a+b);
              break;
         case '-' :
           printf(" %d", a-b);
           break;
         case '*' :
           printf(" %d", a*b);
           break;
         case '\' :
           printf("\n %d", a/b);
            break;
         }
 }
```

Chapter : LINKED LIST.

•

- used to store and organise similar data items in memory.

* Difference b/w LL and an array:

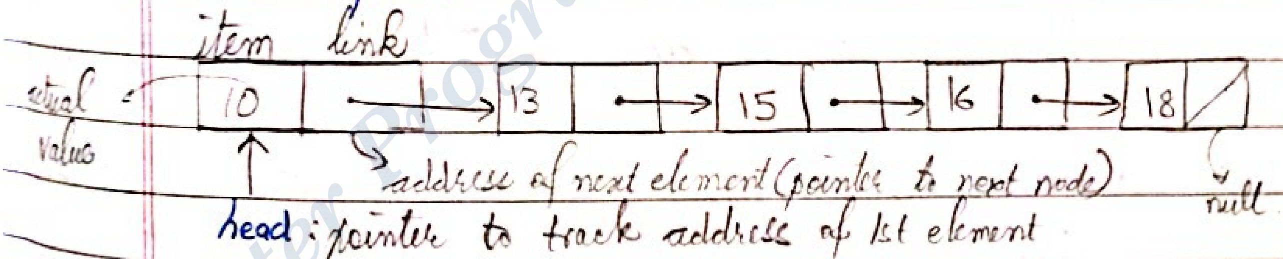
1 Elements of an array are stored in consecutive memory loc^ns.

Elements of a LL are dispersed.

2. In an ordered list of data items, maintained by a LL, insertions and deletions are easier, with no physical data movt.

* Structure of a node in a LL:

Node: an element in a LL. It has 2 parts.


item
link
actual value
10
13
15
16
18
address of next element (pointer to next node)
head: pointer to track address of 1st element
null

* Represent node in a LL as a structure data type:

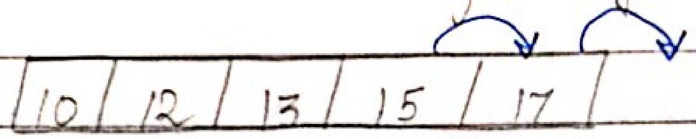
```
struct node
{
    int item;
    struct node * link;
};
```

link field of next node is given by struct node.

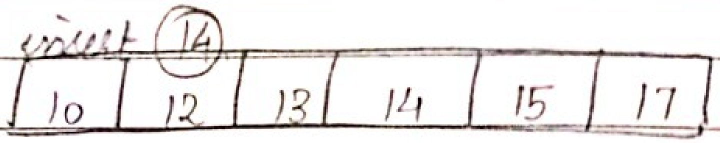
⇓

SELF REFERENCING structures

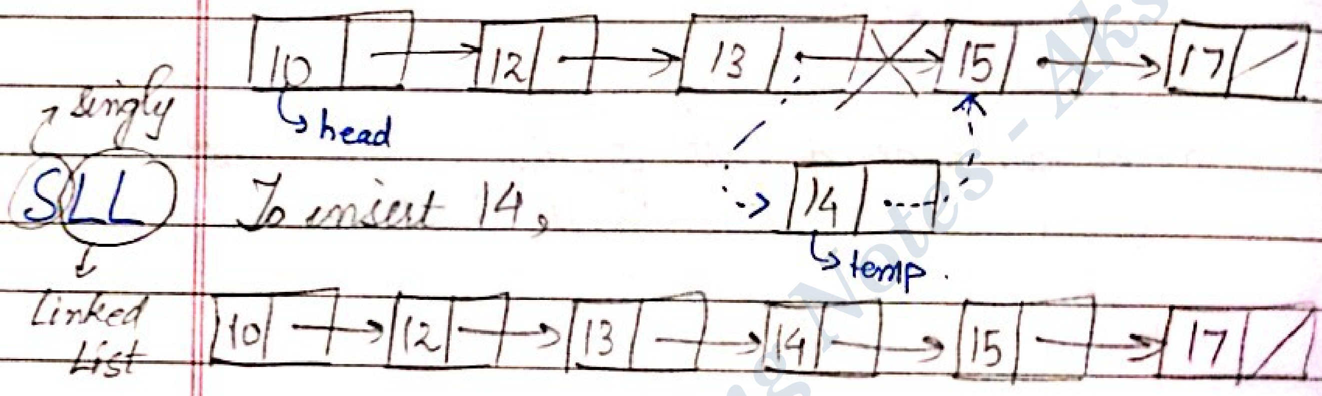
ex An ordered list using arrays.



Now to insert (14)



An ordered list using LL.



To insert 14,

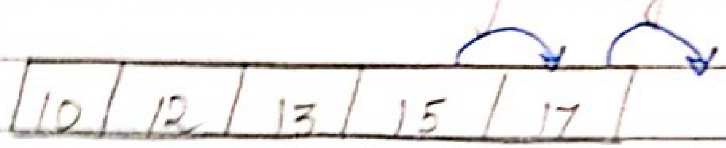
☆ Note:

- Nodes of a LL are created at run time, using malloc()
- head is a pointer to the first node of a LL.
- link field of the last node is NULL.
- if head is known, the LL can be traversed till the end.

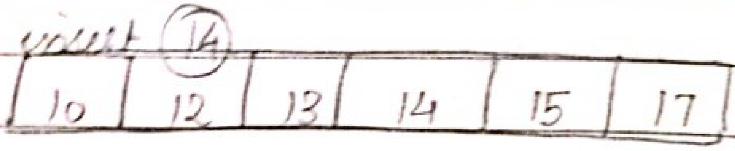
☆ Operations on a LL:

- creating, traversing, displaying a LL.
- Add (insert) a node to the beginning, end or middle of a LL.
- Delete a node from a LL.

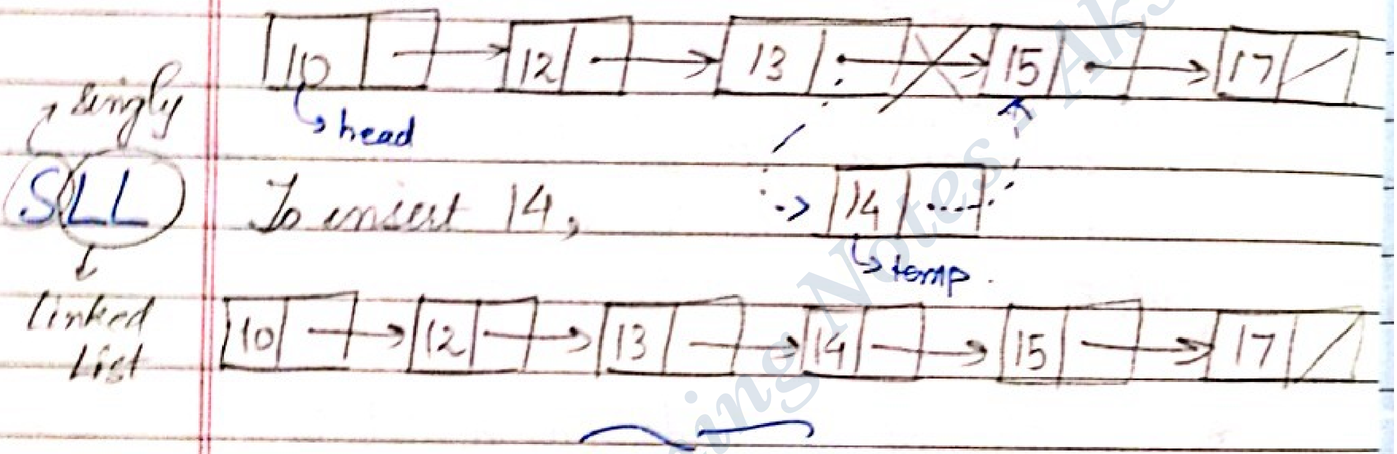
ex. An ordered list using arrays.



Now to insert (14)



An ordered list using LL.



To insert 14,

Note:

- Nodes of a LL are created at run time, using malloc()
- head is a pointer to the first node of a LL.
- link field of the last node is NULL.
- if head is known, the LL can be traversed till the end.

Operations on a LL:

- creating, traversing, displaying a LL.
- Add (insert) a node to the beginning, end or middle of a LL.
- Delete a node from a LL.

* how to check for a list to be empty : Check : head is NULL or not

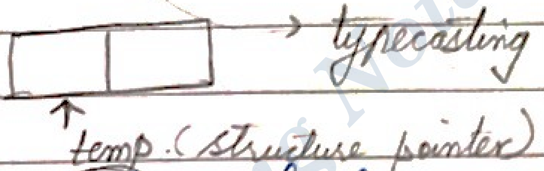
* Other types of LL :
DLL : Doubly Linked List
CLL : Circular Linked List.

* Creating a LL (by appending nodes to a LL

1. Create a new node, temp, using malloc().

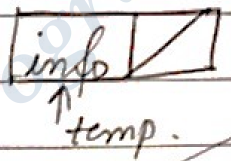
```
temp=(struct node *)malloc(sizeof(struct node))
```

→ typecasting .



2. Read data, info, of the new node.

3. Set the 2 fields of the new node, temp.



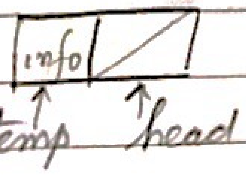
→ accessing each element using arrow operator.

```
temp -> item = info
temp -> link = NULL.
```

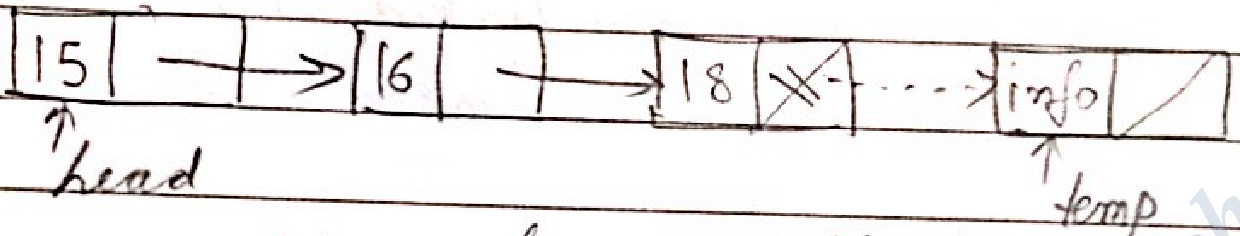
* To append a LL.

4. Check for empty list :- head = NULL.

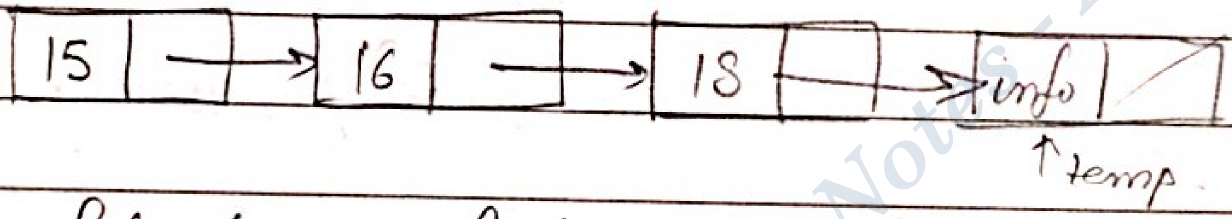
empty list :



non empty list :



Now, traverse till the end



```
while (t -> link != NULL)
        t = t -> link ; /* Move to next no*/
    t -> link = temp;
```

1/

```
* /* creating and displaying an SLL */
#include<stdio.h>
#include<alloc.h>
struct list
 {
  int item;                          { /* structure defin^n */ }
  struct list * link;
  };           - Renaming
  typedef   struct list node;
  main()
   {
   node * head;   /* Pointer to 1st node should be declared*/
   node * create (node *), Pass address of 1st node of LL
   void  display (node *);
    head = NULL;  /* For empty list */
    head = create (head);
    puts ("\n Created list \n");
    display (head);
 }
 /* creating a SLL */
 node * create (node * head)
  {
   node * temp, *t;
   int info;
   printf ("\n Enter data item");
   scanf (" %d", & info);
    while (info != 999)
     {
```

typecasting   allocating space

```
temp = (node *) malloc (sizeof (node));
temp -> item = info ;
temp -> link = NULL ;
  if (head = = NULL)  /* Check for empty list */
      head = temp ;
  else
     {
      t = head ;  /* For traversing a LL, t points to nodes */
      while (t -> link  ! = NULL)
              t = t -> link ;
              t -> link = temp ;
     }
     printf (" \n Enter a data item ");
     scanf (" %d", &info);
    }
 return (head); /* needed for fn display */
}

void display (node * head)
 {
  node * t ;
  t = head ; /* Make t (temp var) point to first node */
  while (t -> link ! = NULL)  or while (t ! = NULL)
   {
    printf ("%d -> ", t -> item);
    t = t -> link ;
   }
  printf (" %d \n", t -> item);   -> Reqd to print the
}                                    last item
```

not required if the above while cond^n is changed.

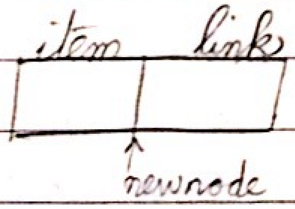
Program: Define a f^n to take address of 1st node & returns the no. of nodes in the LL.

```
/*
 use the main f^n (with some changes)
 from the previous program
*/

int count (node * head)
 { node *t ;
  int c=0 ;
  t = head ;
 while (t != NULL)
    {
        c++
        t = t -> link ;
    }
 return (c) ;
 }
```

☆ INSERTION

1. Create a new node to be inserted.



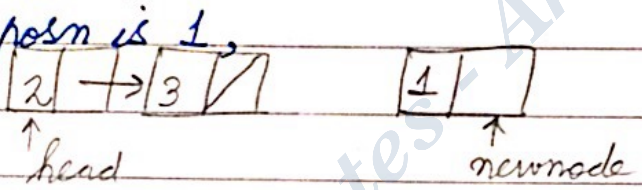
(a) Use malloc() to create newnode
(b) Read data field, data of type new node.
(c) Set the data field of new node
newnode -> item = data

2. Read pos^n of insertion, pos^n
3. Check for empty list.

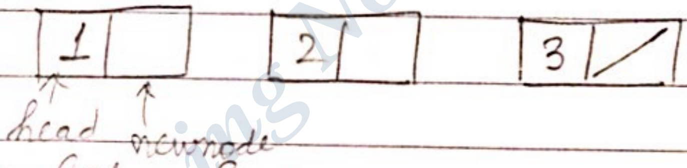
3.1. If list is empty ;
    3.1.1 : Set link field of newnode as NULL.
        newnode -> link = NULL.
    3.1.2 : Make newnode as head
        head = newnode

3.2 If list is not empty,
    3.2.1 : If posn is 1,

Before insertion : (B.I.)



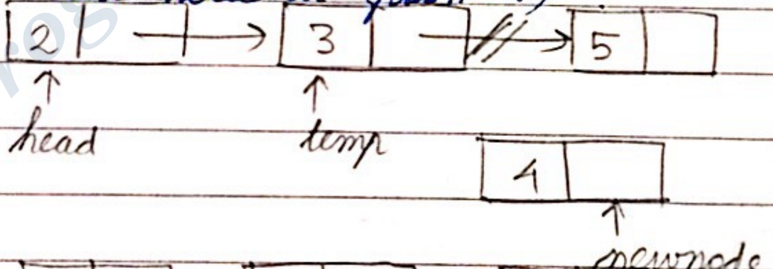
After insertion : (A.I)



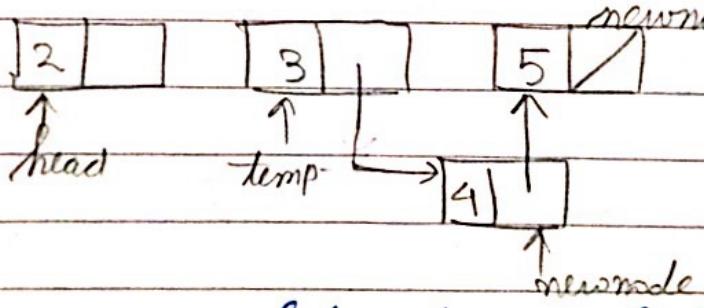
newnode -> link = head ;
head = newnode ;

    3.2.2 : If posn is not 1, traverse and reach the node at (posn-1)

B.I.



A.I.



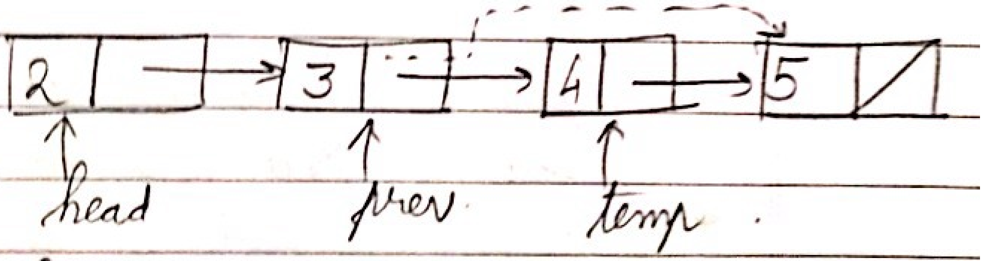
newnode -> link = temp -> link ;
temp -> link = newnode ;

```
node * insert (node * head)
{
  node * newnode, * temp;
  int newitem, i, posn;
  printf("\nEnter item to be inserted\n");
  scanf("%d", & newitem);
  printf("\nEnter position of insertion \n");
  scanf("%d", & posn);
  newnode = (node *) malloc( size of node));
  newnode -> item = newitem;
    if (head == NULL) /*Empty list */
   {
      newnode -> link = NULL;
      head = newnode;
   }
   else /* non empty list */                 not reqd
   {
    if ((posn == 1) && (head != NULL))
    {
      newnode -> link = head;
      head = newnode;
    }
    else
&& temp->link!=NULL     i = 1;
for(i=1; i<posn; i++)   temp = head;
temp = temp         while ((i <= posn-1) && (temp->link != NUL
  ->link;      {
                 temp = temp -> link;
                 i++;
               }
newnode -> link = temp -> link;
temp -> link = newnode;
    }
                    /* Pointer reassignment */
 }
return(head);
}
```

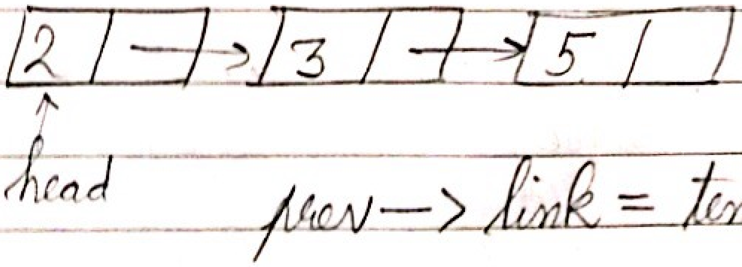
✯ Deletion

1. Read item to be deleted, item
2. If list is empty, print empty list.
3. If list is not empty,
    3.1 : Traverse and reach the node to be deleted (whose item field = item), temp.
    32 : Get the address of previous node, prev.

Before deletion.



After deletion:

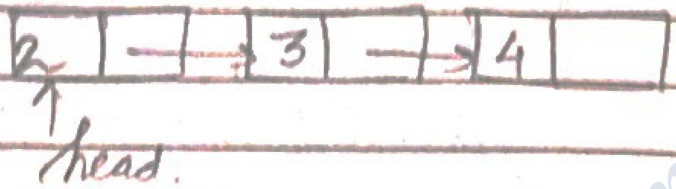


NULL))

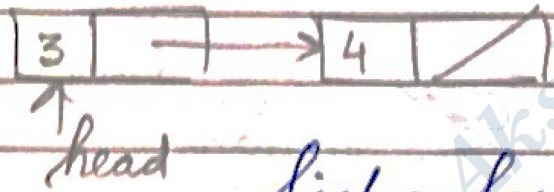
prev -> link = temp -> link

If node to be deleted is the first node

Before deletion item = 2



After deletion



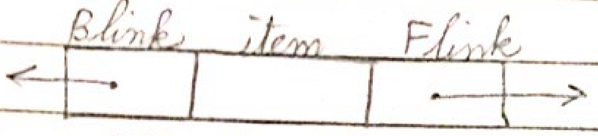
~~first = first~~
head = head->link

—x—

```
node * delete (node * head)
{
node * temp, * prev;
int target;
  if (head == NULL)
          printf("\n Empty list ");
  else
     {
       printf("Enter item to be deleted \n");
       Scanf(" %d", & target);
/* Check whether node to be deleted is
  first node */
   if (head -> item == target)
          head = head -> link;
  else
     {
      temp = head;
        prev = NULL;
while (temp != NULL) && (temp -> item !=
target)).
{
 prev = temp;
 temp = temp -> link;
}
  if (temp = = NULL)
     printf ("\n Element is not found\n");
 else
    {
      prev -> link = temp -> link;
    }
   }
 }
return (head);
}
```

* Doubly Linked Lists (DLL)

- Node:

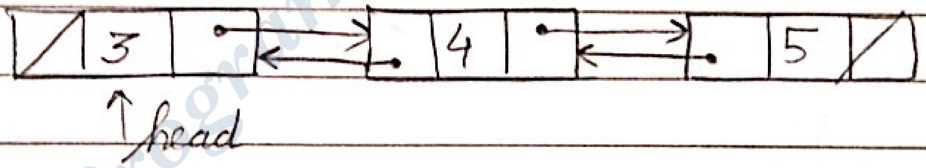


Blink : pointer to the previous node.
Flink : " " " next node
item : actual data.

- To represent node of a DLL:

```
struct list
 {
  struct list *blink;
  & int item;
  struct list *flink;
  };
```

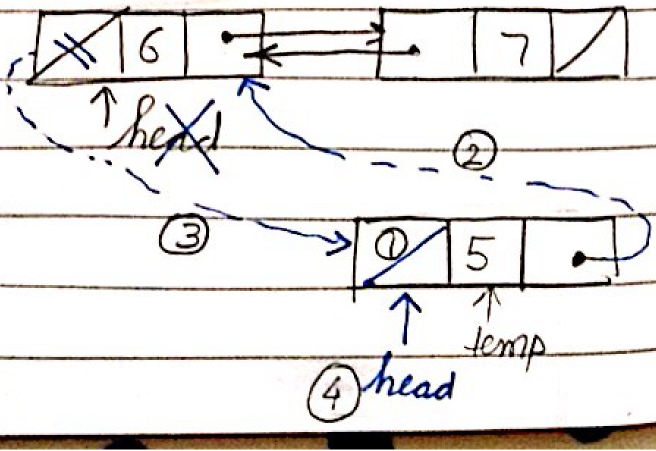
self referencing structure



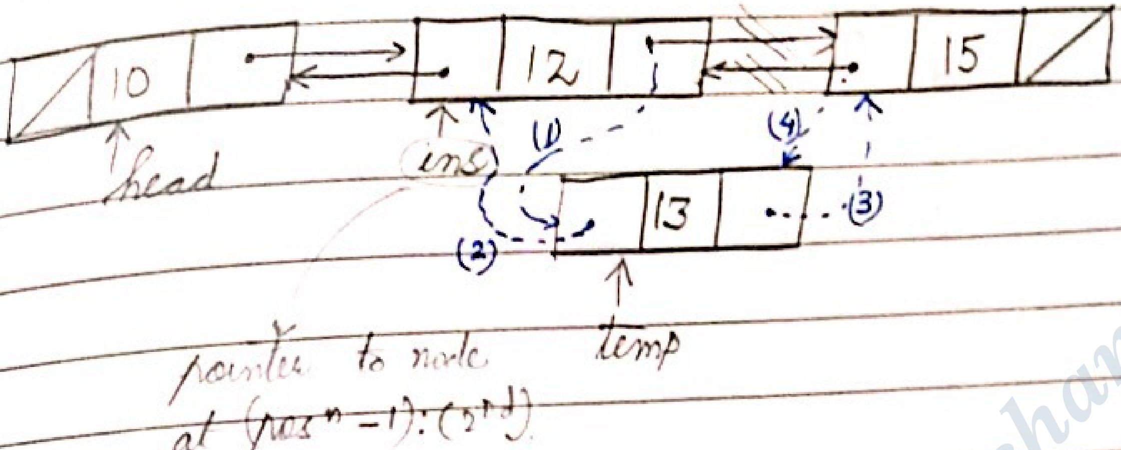
- Insertion :

1. At beginning of a DLL:

Before insertion

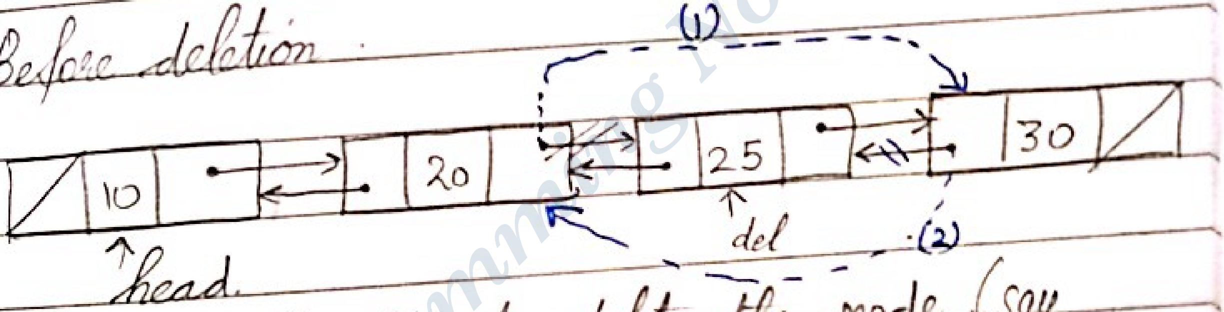


2. At any posn : (say 3rd position).



- Deletion

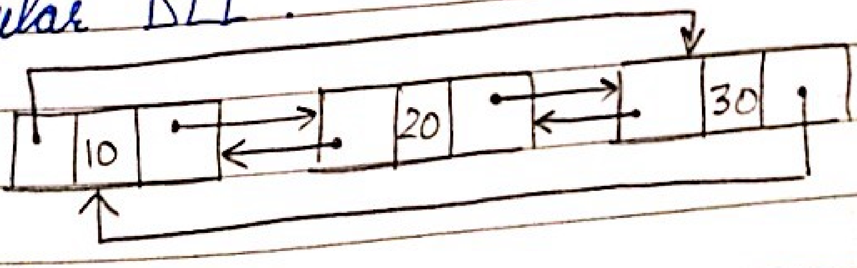
Before deletion :



Traverse the list to delete the node (say node 25)

Circular Linked List :

* Circular DLL.



* Circular SLL