Operators

Operators is a symbol that operate the operands.

Classification - 1

Operators are 3 types

1. Unary operator → One operator operates one operand

   Ex +a , -10 , ! 20

   (type) , & , *

2. Binary operator → One operator operates two operands.

   Ex 5 + 8 , a * b

3. Conditional \ terinary operator → One operator operates three operands.

   exp1 ? exp2 : exp3

   True

   False

There are 45 operators in c language.

Presidence : Which operator will be evaluated first is decided by Presidency. There are 15 Presidency available.

Scanned by CamScanner

Assignment operator (=, presidence no - 14)
Binary operator

```
{
int a=b=c=10;   -> b & c undeclared
printf("%d %d %d", a, b, c);
}
```

error, ~~b & c~~ and

v = v/c/e
b=10

```
a = b;   ✓
a = 3;   ✓
a = b+5; ✓
```

} a is variable

```
3 = a;   ✗
10 = 20; ✗
b+5 = a; ✗
```

} error
L value required

equal(=) operator
right -> Left

```
{
int a, b, c = a = b = 10;
printf("%d %d %d", a, b, c);
}
```

3	5	30 ~~7~~
a	b	c

```
-> {
int a = 3, b = 5, c = 7;
a = b = 20 = c = 30;
printf("%d %d %d", a, b, c);
}
```

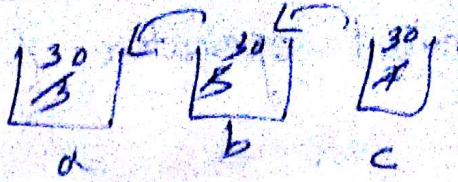
-> L value required

30 can't be stored in 20 as it is a const.

Scanned by CamScanner

→ {

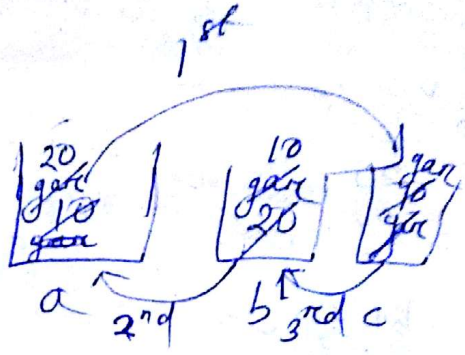
```
int a = 3, b = 5, c = 7;
a = b = c = 30;
printf("%d %d %d", a, b, c);
}
```



O/P : 30 30 30

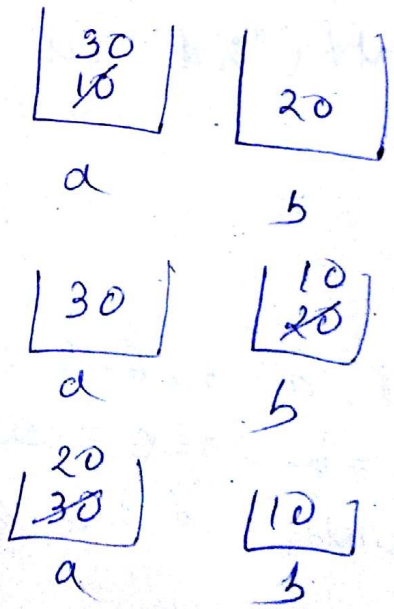
→ Swapping, by 3 variables

```
{
int a = 10, b = 20, c;
c = a;
a = b;
b = c;
printf("%d %d", a, b);
}
```



→ by 2 variables

```
{
int a = 10, b = 20;
a = a + b
b = a - b
a = a - b
printf(  )
}
```



Scanned by CamScanner

-> WAP, take two no from KB, swap it.

```
{
int a,b;
pf("enter two no\n");
sf("%d %d", &a,&b);
printf("before swapping a=%d b=%d\n", a,b);
a = a+b;
b = a-b;
a = a-b;
printf("after swapping a=%d b=%d\n", a,b);
}
```


Scanned by CamScanner

```
{
int a=3, b=5, c=2;
    a=b+c=10;
pf("%d %d %d", a, b, c);
}
```

error

✗ L value required

3, -3, 1, -1

Arithmetic operator

① =

② + - × / %

	10/3 = 3	10%3 = 1
	10/-3 = -3	10%-3 = 1
	-10/3 = -3	-10%3 = -1 } Numerator -ve, result -ve
	-10/-3 = 3	-10%-3 = -1 }
	3/10 = 0	3%10 = 3
	10/0 = Run time error	10%0 = Run time error

```
pf("%d %d %d %d", 10/3, -10/3, 10%3, -10%3)
```

o/p: 3 -3 1 -1

Note

Modulo operator not work in float & double datatype.

```
{ float a = ;
  pf("%f", a%2);
}
```

✗

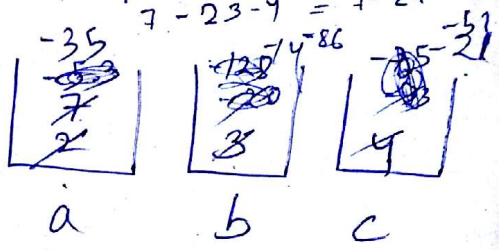
Scanned by CamScanner

```
->{
int a = 10;
  a + 4;  -->  Dummy statement
pf("%d\n", a);
pf("%d\n", a = a*5);
pf("%d\n", a);
}
```

O/P : 10
50
50

```
-> {
int a = 2, b = 3, c = 4;
  a = b + c;
  b = a - 4 * 5 + 3 - c;
  c = a + b * 4 / 2;
  a = b + c;
  c = a + b - 2;
  b = a + c;
pf("%d %d %d", a, b, c);
}
```

a - 20 + 3 - c
~~a - 23 - c~~
7 - 20 + 3 - 4
7 - 23 - 4 = 7 - 27



7 - 20 × 4/2
7 - 80/2
7 - 40
-33

a = -53

c = -53 - 20 - 2
= -75

-53 - 75 = -128

-35 - 14 - 2

O/P: -35 -86 -51

b = -35 - 51
= -86

a = -14 - 21
= 35

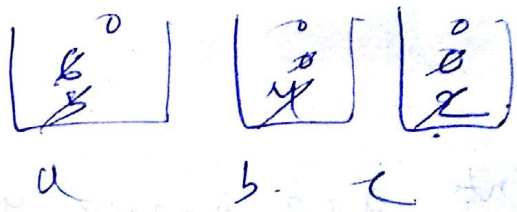
7 - 20 + 3 - 4
-13 + 3 - 4
-10 - 4

7 - 14 × 4/2
7 - 56/2
7 - 28 = -21

Scanned by CamScanner

```
{
int a=5, b=4, c=2;
    a=b+c;
    c=a%3
    b=-c%3*4/2
    a=b+c;
    c= a-b*4%3;
    b= a+c;
pf("%d %d %d", a, b, c);
}
```

O/P : 0 0 0



6/3

b = -0%3*4/2

c = 0-0*4%3

```
{
~~int a=127;~~
int a=127;
    a/10;
pf("%d\n", a);
a = a/10
pf("%d\n", a);
pf("%d\n", a+5);
pf("%d\n", a);
```

Scanned by CamScanner

O/P: 127

12

17

12

→ WAP, take a no from KB. Delete the last digit.

```
{
int a ~~=125~~ ;
 a = ~~a%~~ a/10;
pf("%d", a);
}
```

```
{
 int a;
printf("enter a no\n");
scanf("%d", &a);
a = a/10;
pf("%d\n", a);
}
```

Scanned by CamScanner

→ WAP. Take a no from KB. Display the last digit

```
{
int no;
pf("enter a no\n");
sf("%d", &no);
~~no = no%10~~
pf("last digit = %d\n", no%10);
}
```

→ 2nd ~~long~~ last digit

logic : no/10%10

→ Delete 2nd last digit

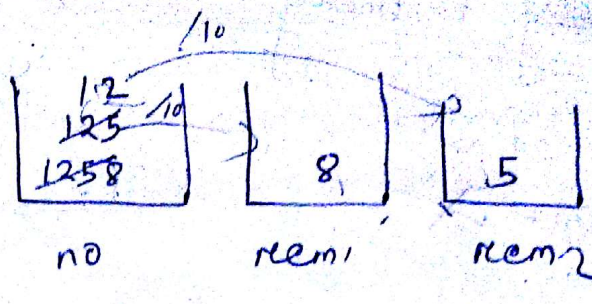
```
{
int no, rem;
pf("enter a no\n");
sf("%d", &no);
rem = no%10;
no = no/100;
no = no*10 + rem;
printf("no = %d\n", no);
}
```

no	rem
257	7

using one variable

no = no/100 x10 + no%10

-> Swap last two digits.




```
{
 int no, rem1, rem2;
 printf("enter a no\n");
 scanf("%d", &no);
 ~~rem1 = 1258/10~~;
   rem1 = no % 10;
   ~~rem 2 =~~
   no = no / 10;
   rem2 = no % 10;
   no = no / 10;
   no = no * 100 + rem1 * 10 + rem2;
   pf("no = %d\n", no);
}
```

-) Using one variable

```
no = no/100 * 100 + no%10 * 10 + no/10 % 10;
```

-> WAP, take a 4 digit no from KB. Display the 1st digit

```
{
int no;
pf("enter a 4 digit no\n");
sf("%d", &no);
no = no / 1000;
pf("no = %d", no);
}
```

Scanned by CamScanner

Unary +, Unary -

+ op ✓

op + ✗

⇒ WAP . .

```
{
 int a = 5;
 + a
 pf("%d\n", a);
 pf("%d\n", -a);
 pf("%d\n", a);
 5
-5
5
```

→ WAP, initialise two integer variable. Add it without using + operator.

```
{
 int a = 5, b = 7, c;
 c = a - (-b);
 pf("c = %d\n", c);
}
```

Scanned by CamScanner

incredment ++ } Unary
Decrement --

++a; ~~= a+~~
a = a+1

a++
a = a+1

increment
- Pre: ++op
- Post: op++

Decrement
- Pre: --op
- post: op--

-> Increment, decrement operator not applicable to constant & expression. It is only applicable to variables.

```
{
  int a = 5;
  ++10;
  ++(a+3);
}
```

Error
L value required
10 = 10+1
= 11 ✗

-> If you write simple ++a or a++, meaning is same.

```
{
  int a = 5;
  a++;
  pf("%d\n", a);
}
```

op : 6

++a

op : 6

Scanned by CamScanner

```
{
int a=5, b;
b = a++;
pf("%d %d\n", a,b);
}
```

O/P : 5 6

a: 6 ~~5~~ b: 5

b = a

a = a+1

First stores value then increase

```
{ int a=5, b;
  b = ++a;
  pf("%d %d\n", a,b);
}
```

1st increases then stores

O/P : 6 6

```
{
int a=5, b;
b = a++ + a++ + a++;
pf("%d %d", a, b);
}
```

O/P : 8 15

```
b = a-- + a-- + a--;
pf("%d %d", a, b);
```

O/P : 2 15

Scanned by CamScanner

```
-> {
   int a=5,b;
   b = ++a + ++a + ++a
       6   7   7   8
   pf("%d %d", a,b);
   }
```

O/P: 8 22

```
-> {
   int a=5,b;
   b = --a + --a + --a;
       4   3   3   2
   Pf("%d %d", a,b);
   }
```

O/P: 2 8

```
-> {
   int a=5,b;
   b = a++ + ++a + a++ + ++a;
       5   6   6   6   7
   pf("%d %d", a,b);
   }
```

After semicolon a value increases due to post increment

O/P : 9 25

Scanned by CamScanner

int a=5, b;

b = a++ + a b = ++a + a b = a++ + a++

O/P: 6 10 6 12 7 10

b = ~~a++~~ ++a + ++a b = a++ + ~~a++~~ ++a

6 7 5 6

7 14 7 12

-> Increment & decrement are applicable to all ~~data~~ primitive datatype variable.

```
{
int a = 5;
~~ft = 12.34~~
float ft = 12.34f;
char ch = 'y';
++a ; ++ft ; ++ch;
printf("%d ~~%d~~ %f %c\n", a, ft, ch);
}
```

o/p: 6 13.34000 z

```
{
int a = 5;
pf("%d\n", a);          O/P = 5
pf("%d\n", a++);        5
pf("%d\n", --a);        5
pf("%d\n", a--);        5
pf("%d\n", ++a);        5
pf("%d\n", a);          5
}
```

Relational operator

1 == 0 != 0

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

All relational operator always return true for 1, false for 0.

2 < 3 1

2 < 3 < 4 1

4 > 3 1

4 > 3 > 2 0

3 <= 3 1

3 <= 2 0

4 >= 4 1

3 >= 4 0

3 == 3 1

3 == 4 0

3 != 4 1

3 != 3 0

3 == 3 != 0 0

2 < 3 == 3 > 4 != 5 < 2

1 == 0 != 0

0 != 0

0

Scanned by CamScanner

```
int a = 2, b = 4, c = 5;
c = a+b < 2 > = 5;
a = c + 4 % 3;
b = - a % 3 / 2;
++a;
b--;
-a;
pf("%d %d %d %d \n", a > b, a + 3, b, c);
}
```

6 < 2 > = 5
0 > = 5
0

a	b	c
2	-1	0

O/p : 1 5 -1 0

4 % 3

-1 % 3 / 2
-1 / 2
0

```
{
int a = 5;
pf("%d %d %d", a, a > 5, a = 6);
}
```

6 1 6

~~O/p : 5 0 6~~

6

O/p : 6 1 6

Logical / short ckt operator

|| } Binary
&& } Binary
! } Unary

→ All logical operator always return true for 1 & false for 0.

→ In C language all non zero value is known as true.

Op 1	Op 2	result
T	T	T
T	F	T — 1
F	T	T
F	F	F — 0

→ If 1st operand is true, then 2nd operand isn't checked.

~~→ If 1st~~

```
{
  int a=5, b;
  b=10 || ++a;
  Pf("%d %d", a,b);
}
```

~~b=a++~~

As 1st operand is non-zero ++a isn't checked

O/P : 5 1

Scanned by CamScanner

```
-> {
   int a=5, b;
   b=0||++a;
   pf("%d %d", a, b);
   }
```

O/P: 6 1

As 1st operand is 0, 2nd oper. is checked

~~6=1~~

~~6=7~~

```
-> {
   int a=5, b=7, c;
   c = ++a || ++b;
   pf("%d %d %d", a, b, c);
   }
```

O/P: 6 7 1

```
-> {
   int a=0, b=0, c;
   c=a++||b++;
   pf("%d %d %d", a, b, c);
```

O/P: 1 1 0

Scanned by CamScanner

Logical ~~&~~ and (&&)

OP 1	OP 2	result OP3
T	T	T → 1
T	F	F
F	T	F → 0
F	F	F

→ If 1st operand is false, 2nd operand isn't checked.

```
{
 int a=5, b;
 b=10 && ++a;
pf("%d %d", a, b);
}
```

O/P : 6 1

→
```
{
 int a=5, b;
 b=0 && ++a;
pf("%d %d", a, b);
}
```

O/P : 5 0

Scanned by CamScanner

```
}
int a=1,b=0,c;
  c=a-- && b--;
pf("%d %d %d",a,b,c);
}
```

1 && 0

0 -1 0

O/P : 0 -1 0

Logical not (!=)

!a ✓

a! ✗

Op	result
T → Any non zero value	F — 0
F → Zero	T — 1

```
{
int a=7,b;
b=!a;
a!=b;      → Non-updatable st.
pf("%d %d\n",b,a);
pf("%d %d\n",!b,!!a);
pf("%d\n",a);
}
```

7	0
a	b

O/P : 0 7

1 1

7

Scanned by CamScanner

Bitwise operator

|	Bitwise or
&	Bitwise and
^	(XOR)
<<	left shift
>>	right shift
~	Complement

29.05.17

Bitwise operator doesn't work in float & double datatype.

1st convert decimal no binary. Then operate these operator. Again convert to decimal.

|		
1	1	1
1	0	1
0	1	1
0	0	0

&		
1	1	1
1	0	0
0	1	0
0	0	0

^		
1	1	0
1	0	1
0	1	1
0	0	0

```
{
int a = 6, b=10, c;
c = a|b;
pf("%d\n", c);
c = a&b;
pf("%d\n", c);
c = a^b;
pf("%d\n", c);
```

	16	8	4	2	1	
		0	1	1	0	
		1	0	1	0	
O/P						
14	1	1	1	0		|
2	0	0	1	0		&
12	1	1	0	0		^

Scanned by CamScanner

```
->{
int a=23,b=17,c;
c= a|b;
pf("%d\n", c);
c= a&b;
pf("%d\n", c);
c= a^b;
pf("%d\n", c);
}
```

	32	16	8	4	2	1	
		1	0	1	1	1	
		1	0	0	0	1	
23 —		1	0	1	1	1	1
17 —		1	0	0	0	1	&
6 —		0	0	1	1	0	^

O/P : 23
17
6

Left shift (<<)

```
{
int a=5,b;
b = a<<1;
pf("%d\n", b);
}
```

16	8	4	2	1	
0	0	1	0	1	
0	1	0	1	0	— 10

O/P: 10

or multiply by 2 for every 1 shift.

b = a<<3;

5 x 2 = 10
10 x 2 = 20
20 x 2 = 40

Scanned by CamScanner

→ How many shift in left side that much zero add in right side.

```
{
int a=5, b;
b = a<<3;
pf("%d\n", b);
}
```

5

32 16 8 4 2 1

1 0 1 0 0 0

O/P: 40

Right shift

```
{
int a=10, b;
b = a>>1;
pf("%d\n", b);
}
```

O/P: 5

8 4 2 1

1 0 1 0

1 0 1 → 5

1 0 1 0

1 shift

→ Divide by 2 for every 1 shift. How much shift in right side delete that much digit from right side.

1 0 1 0

1 shift

delete 0.

101 → 5

Complement

+ve no. is stored in the computer directly in the binary format but -ve no is stored in the computer in 2s complement format

Scanned by CamScanner

2s complement = 1s complement + 1

+ve
(5) is stored in computer in this way

Sign bit ~~data~~ ~~datatype~~ bit
5 31

$\boxed{0}$ (As +ve) 0 0 0 28 times — 1 0 1

-ve no is stored
(-6)

Sign bits data bits

$\boxed{1}$ (As -ve) 28 times 0 0 0 0 1 1 0

28 times 1 1 1 1 1 0 0 1

+1

~~Conve~~ (By 2s compliment)

$\boxed{1}$ 1 1 1 1 1 1 0

(+ve no) -> ~ is a unary operator

```
-> {
  int a=5, b;
  b = ~a;
  pf("%d %d \n", a, b);
}
```

$\boxed{0}$

Scanned by CamScanner

2 {

~~int a = 5~~

	Sign	data
(+ve)	$\boxed{0}$	00 ... 000101
1st sign bit changes	$\boxed{1}$	1s complment 11 ... 111010
when sign value changes again 2s complment is done ~~& 1 is added~~.	$\boxed{1}$	again 2s complment 00 ... 000101 + 1 00 ... 000110

OR

increase by 1 and assign -ve sign.

(-ve no)

```
{
  int a=-5, b;
  b=~a;
  pf("%d %d\n",a,b);
}
```

$\boxed{1}$	00 ... 000101
$\boxed{1}$	11 ... 111010
	1
$\boxed{1}$	11 ... 111011
$\boxed{0}$	00 ... 000100

1st store the -ve no.

Then apply 1s complment.

O/P: -4 = -5 + 1.

Scanned by CamScanner

SWAP initialise two integer variables

Substract 1st no to from 2nd no without operation

```
{
int a = 20, b = 11, c;
c = a+~b+1;
pf("%d\n",c);
}
```

$20 - 12 + 1$
$= 9$

Add without using + operation

$c = a - (\sim b) - 1$

Compound assignment operation

$+=$
$-=$
$*=$
$/=$
$\%=$
$|=$
$\&=$
$\wedge=$
$<<=$
$>>=$

$a += 3$
$a = a + 3$

$a \wedge = b$
$a = a \wedge b$

It is evaluated right to left.

Scanned by CamScanner

```
int a = 2, b = 3, c = 4,
c += a;
a *= 3;
b -= c;
a <<= 1;
b *= 2;
c = a ^ b;
pf("%d %d %d", a, b, c);
}
```

O/P: 12 -6 -10

```
{
int a = 2, b = 3, c = 4;
a += b * = c - = 2;
pf("%d %d %d\n", a, b, c);
}
```

$\cancel{a = a + b} = a + b * c = \underline{a + b * c - 2}$

$a + +2$

$c = c - 2$

$b = b * c$

$a = a + b$

O/P: 8 6 2

Scanned by CamScanner

```
{
  int a = 2, b = 3, c = 4;
  a *= b+c;
  pf("%d\n", a);
}
```

a = ~~14~~ 2

b+c = 7

a = a*7

O/P : 14

Special operator

( ) → Highest presidence

, → Lowest "

(type)

sizeof

?:

```
{
  int a = 2, b = 5, c;
  c = (a, b, 20);
  pf ~~("%d\n", c);~~ ("%d %d\n", a, c);
}
```

O/P: 10 20

Note

→ Rightmost value is stored in c if ( ) is used.

Scanned by CamScanner

Comma operator

- Last precidence
- can convert multiple line to single line
- Evaluated left to right

```
{
 int a=2, b=5, c;
 c = a, b, 20;
 pf("%d\n", c);
}
```

O/P : 2

(type) operator -> binary operator

```
{
 int a = 5, b=2;
 float c;
 c = a/b;
 pf ("%f\n", c);
 c=(float) a/b;
 pf ("%f\n", c);
}
```

-> a is explicitily converted into real no. i.e. 5.0

So 5.0/2 = 2.500000

O/P : 2.000000

2.500000

Scanned by CamScanner

Sizeof operator

Sizeof is an operator, is used to measure how much memory occupy a variable or datatype & expression.

It isn't applicable to function.

It is a unary operator.

→ {

```
char ch='y';
pf("%d\n", sizeof ch);
pf("%d\n", sizeof (12.34));
pf("%d\n", sizeof (int));
}
```

O/P: 1

8

4

→ {

```
int a=10;
pf("%d\n", sizeof a+4);
pf("%d\n", sizeof a+4.0);
}
```

O/P: 4

8

Scanned by CamScanner

sizeof (4+a)

O/P: 4

```
-> {
   int a=10, b;
   b=sizeof(++a);
   Pf("%d %d\n", a, b);
 }
```

O/P: 10 4

Not 11, because inside sizeof there is no manipulation.

Tereinary operator / Conditional operator

exp1 ? exp2 : exp3 Presidence 13

if True

If false

```
{
printf("%d\n", 10 ? 20 : 30);
}
```

O/P -> 20

Scanned by CamScanner

```
-> {
    int a=5, b;
    b= a>4 ? 20: 30;
    pf("%d\n", b);
  }
```

o/p -> 20

```
->
  {
    int a=5, b;
    a>4 ? b=20 : b=30;
    pf("%d\n", b);
  }
```

error: L value required

```
->
  {
    int a=5, b;
    a>4 ? (b=20) : (b=30);
    pf("%d\n", b);
  }
```

o/p -> 20

-> return 10? 20: 30; ✓

10? return 20 : return 30; ✗

10? return 20: 30; ✗

Scanned by CamScanner

In pointer we can use 2 operators

* : Dereferencing / Indirectional / Value at that address

& : Address operator

→ In Array

[ ] subscript operator

→ . } Member acess operator

→ - > } ~~St~~ Used in str and union.

Unary

+	*
-	&
++	
--	
!	
~	
sizeof	
(type)	

Total 10 unary operators

? :] ternary

Rests are in binary

Scanned by CamScanner

Input & Output function

Unformating	Formating
getch( ) - not eco	printf( )
getche( ) - eco	scanf( )
getchar( ) - eco + enter	
gets( )	
putch( )	
putchar( )	
puts( )	

getch(), getche(), getchar()

-> These are designed to take a character from KB.

```
{
  char ch;
    pf("enter a ch\n");
    ch = getch();      // ch = getche()      // getchar()
       // eco not        // eco                // eco + enter
    pf("\n ch = %c", ch);

}
```

Scanned by CamScanner

putch Display character

```
{
char ch='y';
putch(ch);
putch('\n');
putch('a');
}
```

O/P: y
a

- putch & putchar are same
- It is an output function.
- Takes one argument which must be character type.
- putch display which character into the moniter that character's Aski value is returned.

```
{
char ch='a';
int b;
b= putch(ch);
pf("~~%d~~ \n %d", b);
}
```

O/P: a
97

puts()

- puts() is a string unformating output function.
- It takes only one arguement that must be string type.

Scanned by CamScanner

```
{
 char str[20] = "bbsr";
 puts(str);
 puts("purci");
}
```

O/P: bbsr
purci

→ puts() automatically breaks line.

gets()

→ Unformating input function.
→ Take a string from KB.

```
{
 char str[20];
 puts("enter a streing");
 gets(str);
 puts(str);
}
```

Scanned by CamScanner

Formating

-> printf

-> is a formating output function.

-> Display how many characters into the monitor, that much value is returned. So it's is returned type ~~function~~ is integer type.

-> Always evaluated from right to left but substitute the value left to right.

-> You can use it in two ways.

1. Message oriented print fun^n.
   It takes only one arguement that must be string type.

2. Result oriented
   It takes more than one arguement
   1st arguement must be string type
   others may be variable, const, expression, fun^n . . . .

```
{
pf("%d one", pf("two"));
}
```

two 3 one

Scanned by CamScanner

→ {

int a = 2378;

pf("%d\n", a); —— 2378 ~~2378 2734~~

pf("%10d\n", a); — 2378

width

pf("%010d\n", a); — 0000002378

pf("%-10%dhi\n", a); - 2378000000

pf("%2d\n", a); — 2378

}

→ Rule isn't applicable as width no. is less than digits in given no.

→

{

int a=23, b=56;

pf("%5d %4d\n", a, b);

}

O/P: - - - 23 - - 56

a

b

13.4
w.f

```
->{
  float a = 12.34;
  pf("%f\n",a);          ——— 12.340000
  pf("%13f\n",a);        ——— 12 340000
  pf("%013f\n",a);       ——— 0000 12 340000
  pf("%-13f hi\n",a);    ——— 12.340000 hi
  pf("%.4f hi\n",a);     ——— 12.3400
  pf("%13.4f\n",a);      ——— 12.3400
  pf("%013.4f\n",a);     ——— 000000 12.3400
  pf("%-13.4f hi\n",a);  ——— 12.3400 000000 hi
  pf("%3f\n",a);         ——— 12.340000
}
```

```
->
  char ch='y';
  pf("%c\n",ch);         ——— y
  pf("%10c\n",ch);       ——— y
  pf("%010c\n",ch);      ——— 0 000000000y
}
```

Scanned by CamScanner

{

char str[20] = "lakshya";

pf ("%s\n", str); — lakshya

pf ("%13s\n", str); — lakshya

pf ("%013s\n", str); — 00000 lakshya

pf ("%-13shi\n", str); — lakshya ..... hi

pf ("%.4s\n", str); — laks

pf ("%13.4s\n", str); — ..... laks

pf ("%013.4s\n", str); — 00000000 laks

pf ("%-13.4shi\n", str); — laks ..... hi

pf ("%3s\n", a); — lakshya

}

1. (b) -1

2. (b) -1

3. 1

4. -1

5. 24

6. (d) ~~Modulo~~ % not work in float & double datatype

7. 1

8. a

9. (c) a!=10 logical

10. (d) float

Scanned by CamScanner

11 (b) 1

$a == (b + c)$

$10 == 10$

1

12. (a) a 2 b 1

$d = 1$

13. printf("%d %d %d", ++a + ++a + a++ . a++ + ++b, ++d + d++ + a++)

2 2 2 2 1

a b c

a b d

(c) 9 3 5 (a) 15 4 5

14. (a) 0

$0 + 0$

15 (a) 6

16. (a) 0 2

17. (a) 97

18. (c) -5

19. (a) syntax error

20. (b)

21. (b) 2.5

22. (c) Reserved words

23. (b) main( )

24. (b) format specifier

25. (b) High level

26. int z = y && (y/=10);

(b) 0

$y = y/10$

$0/10$

$0 \&\& 1$

27. 1 (right most value is stored)

Scanned by CamScanner

28. (b) 1

29. (b) 20

30. (b) 1

31. z = x && y = 1

z = 2 && 0 = 1

0

(c) Compile time error

32. 2 + 1 + 3 - 5

6

(b) 6

33. (b) 3

34. (b) - 2

35. a = 2 + 2

(d) 14

36. (b) 9

37. 00000

32. 4

38.

39. (b) 15

40. (c) 9, 10

Scanf

->

-> Scanf fun^n take how many value take from KB that much value return.

-> Scanf fun^n return type is integer type.

-> scanf fun^n 1st arguement must be string type & that string contain only format specifier.

-> Other arguement variable with address operator

```
-> {
   int a,b,c;
   c = scanf("%d %d", &a, &b);
   pf("%d %d %d", a,b,c);
   }
```

O/P -> garbage garbage 2.

```
-> {
   int a,b,c;
   pf("enter two nos\n");
   sf("%3d %2d", &a, &b);
   pf("%d %d", a, b);
   }
```

O/P -> 23456
234 56

25 3525
25 35

3427
342 7

Scanned by CamScanner

```
{
float a;
pf ("enter a real no\n");
sf ("%4f", & a);
pf ("%f ", a);
}
```

O/P -> 12.3456

12.300000

```
{
char str[20];
pf ("enter a string \n");
sf ("%3s", str);
pf ("%s ", str);
}
```

O/P -> lakshya

lak

```
{
int a, b, c;
pf ("enter three nos\n");
sf ("%d %d %d ", & a, & b, & c);
pf ("a = %d  b = %d  c = %d", a, b, c);
}
```

O/P -> a = 10 b = 20 c = 30

O/P -> 10 30 garbage

Scanned by CamScanner

→ Two ship any no, put * betn % & d

→ WAP. take two nos from KB & display it in 2 statements.

```
{
 int a, b;
 pf ("a = %d b = %d "; a, b, scanf ("%d %d", &a, &b);
      pf ("enter two nos\n"));

}
```

→

```
{
 int a;
 a = pf ("byebye %d ", pf ("hi okok\n"));
 pf ("%d\n", a);
}
```

O/p → ~~hi okok~~ byebye 8 7

→

```
{
 int a;
 a = pf ("hi") + pf ("okok") * pf ("haha");
 pf ("%d\n", a);
}
```

O/p: hi okokhaha 18

Scanned by CamScanner

Ch 5

Control ~~struct~~ statement

Sequential control str.

-> Every program must be executed line by line

-> We can't skip any statement. That is known as sequential control str.

Previous all programs are sequential control str. type.

-> In this type of program there are no if, if else, for, while, do while etc.

e.g.

```
{
 printf("A");
 pf("B");
 pf("C");
}
```

Conditional ctrl str

- if -> only if
  if else
  else if -> else if ladder

Scanned by CamScanner

if

```
{ printf ("A");
  if (3<5)
  {
    pf ("B");
  }
  pf ("c");
}
```

→ if with body

```
if (condition)
{
  if true stmt.
}
```

→ {

```
int a=4, b=5;
pf ("A");
if (a==b)
{
  printf("B");
}
pf("c");
}
```

O/P → A

c

Scanned by CamScanner

→ WAP. Take a no from KB. check no is +ve no.

```
{
 int no;
 pf("enter a no\n");
 sf("%d", & no);
 if (no >=0)
 {
  pf("+ve no");
 }
}
```

→ WAP, take a no from KB. check no is divisible 3.

```
{
 int no;
 pf("enter a no\n");
 sf("%d", & no);
 if (no%3 == 0)
 {
  pf("no is divisible by 3");
 }
}
```

Scanned by CamScanner

If without bodypart

→ if we don't provide { } in if block, the compiler provide~~d only~~ one statement to { }, to one statement.

```
{
pf("A");
if(3<5)
pf("B");
pf("c");
}
```

Compiler converts

```
{
pf("A")
if (3<5)
{
pf("B");
}
pf("c");
}
```

This rule is also applicable ~~if~~ to if else, else if ladder, switch, while, do while, for.

~~If~~ if with ;

```
pf("A");
if (3>5);
pf("B");
pf("c");
```

Compiler converts

```
{
pf("A");
if (3>5)
{
;
}
pf("B");
pf("c");
}
```

O/P → ABC

Scanned by CamScanner

→ If we execute multiple statement inside if block, then the programmer must ~~be~~ provided { }.

→ WAP, take an employee salary from KB. If salary is >= 5000, ta = 20%, da = 25%. Then calculate the total salary.

5000 * 20 / 100

5000 * 100

```
{
 float sal, ta=0, da=0;
 pf("enter a sal\n");
 sf("%f", &Sal);
 if (sal >=5000)
 {
   ta = sal * .2;
   da = sal * .25;
 }
 pf("total sal = %.2f\n", sal + ta + da)
}
```

Multiple if

→ More than one if in a program, that is known as multiple if.

→ Every if condition must be checked

Scanned by CamScanner

```
-> {
   pf ("A");
   if (3>5);
   pf ("B");
   if (10<20)
   pf ("C");
   pf ("D");
 }
```

O/p → ABCD

```
-> {
   pf (A");
   if (3>5)
   pf ("B");
   if (10<20)
   pf ("C");
   pf ("D");
 }
```

ACD

-> WAP, take two nos KB. Check both nos are same.
check 1st no is greater than 2nd no
check 2nd no is greater than 1st no

```
{
  int no1, no2;
  pf ("enter two nos \n");
  ~~if (no1 no2)~~ sf("%d%d", &no1, &no2);
  if (no1 = = no2)
```

Scanned by CamScanner

```
{
 pf("no1 and no2 are same\n");
}

if (no1 > no2)
{
 pf("no1 is greater than no 2\n");
}

if (no2 > no1)
 {
 pf("no2 is greater than no1\n");
 }
}
```

-> WAP, take a no from KB. check no is divisible by both 5 & 7.

```
{
 int a;
 pf("enter a no\n");
 sf("%d", &a);
 if(a%5==0 && a%7==0);
 //(a%5==0)
 {
  pf("a is divisible by both 5 & 7");
 }
}
```

Scanned by CamScanner

Nested if

-> Nested if & multiple if both are not same

-> In nested if, If outer if is false inner if isn't checked.

but in multiple if, all if is checked.

```
int a=2, b=5;
if(a==2)
{
    printf("A");
    if(b==4)
    {
      printf("B");
    }
  printf("C");
}
printf("D");
}
```

O/P -> A C D

-> WAP, take a no. from KB. check no is 2 digit using nested if.

```
int no;
pf("enter a no\n");
sf("%d", &no);
if(no>9)
{
  if(no<100)
```

Scanned by CamScanner

```
    {
    pf ("no is 2 digit\n");
    }
}
}
```

> if (no>9 && no<100)

```
if (no>9)
if(no<100)
pf("no is 2 digit\n");
```

Compile convert

```
{
  if(no>9)
  {
  if (no<100)
  {
   pf ("no is 2digit\n");
   }
}
}
```

Scanned by CamScanner

If else

```
if ( condition )
{
  True block
}
else
{
  false block
}
```

True

False

```
-> {
    pf("A");
    if (3<5)
    pf ("B");
    else
    pf ("C");
    pf ("D");
  }
```

O/P -> ABD

```
-> {
    pf ("A");
    if (3<5)
    pf ("B");
    pf ("C");
    else
    pf ("D");
    pf ("E");
  }
```

O/P -> Error

'else' without previous 'if'.

Scanned by CamScanner

```
{
  pf ("A ");
  if (3 > 5)
  {
    pf ("B");
    pf ("C");
  }
else
{
  pf ("D");
  pf ("E ");
}
}
```

O/P -> A D E

```
{
  int a = 5;
  if (a = = 5)
  {
    a = 20;
  }
  else
  {
    a = 40;
  }
pf ("%d\n", a);
}
```

O/P -> 20

Scanned by CamScanner

→ int a=5;

```
if (a=0)        //if (a==5) → O/P → 20
{
  a=20;
}
else
{
a=40
}
pf("%d\n", a);
}
```

O/P → 40

a=0, i.e. false

→ WAP, take a no from KB. check no is +ve or -ve.

```
{
int no;
pf("enter a no\n");
sf("%d", &no);
if (no >=0)
{
 pf("no is +ve\n");
}
else
{
 pf("no is -ve\n");
}
}
```

Scanned by CamScanner

→ WAP. take no from KB. check no is three digit.

```
{
    int no;
    pf("enter a no\n");
    sf("%d", &no);
    if (no >99 && no < 1000)
    {
      pf("no is three digit\n");
    }
  else
    {
      pf("no is not three digit");
    }
}
```

→ WAP, take an employee salary from KB. If salary is greater than equal 5000, ta = 20%, da = 25%. if salary is less than 5000 ta = 10%, da = 10%. Calculate total salary.

Scanned by CamScanner

```
{
float sal;
pf("enter a sal\n");
sf("%f", &sal);
if (sal>=5000)
 {
  ta = sal * .2;
  da = sal * .25;
 }
else
 {
  ta = sal * .1;
  da = sal * .1;
 }
pf("total sal = %f\n", sal+ta+da)
}
```

—x—

31.05.17

Nested if else

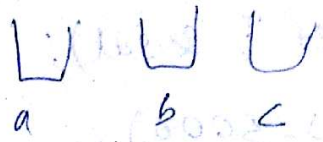
```
if(condition1)
{
   if(condition 2)
   {
   }
}
else
{
   if(condition 3)
   {
   }
   else
   {
   }
}
```

True

False

Scanned by CamScanner

→ ~~Wai~~ WAP, take 3 nos from KB. Display the biggest no.




```
{
int a,b,c;
Pf("enter 3 nos\n");
sf("%d%d%d", &a,&b,&c);
if (a>b)
 {
   if (a>c)
   Pf("a is biggest no\n");
   else
   pf("c is biggest no\n");
 }
else
 {
   if (b>c)
   pf("b is biggest no\n");
   else
   pf("c is biggest no\n");
 }
}
```

Scanned by CamScanner

```
-> {int a=1, b=2, c=3;
   if (a==2)
   if(b==2)
   if (c==3)
   pf("hi\n");
   else
   pf("byebye\n");
   pf("haha\n");
   }
```

```
{
int a=1, b=2, c=3
if (a==2)
 {
   if (b==2)
   {
    if (c==3)
    {
     pf("hi\n");
    }
    else
    {pf("byebye\n");
    }
   }
 }
pf("haha\n");
```

O/p -> haha

```
-> {int a=1, b=2, c=3;
   if (a==1)
   if(b==2)
   if (c==3)
    pf("hi\n");
   else
   pf("haha\n");
   else
   pf("haha\n");
   else
   pf("haha\n");
   }
```

```
{
int a=1, b=2, c=3;
if(a==1 && b==2 && c==3
 {
  pf("hi\n");
 }
else
 {
  pf("haha\n");
 }
}
```

O/p -> ~~haha~~ hi

Scanned by CamScanner

→ WAP, take a character from KB. check the character is alphabet or not.

```
{
 char ch;
 pf ("enter a ch\n");
 ch=getche();
 if ((ch>='A' && ch<='Z')||(ch>='a'&& ch<='z'))
 {
   pf ("\nalp");
 }
 else
 {
   pf ("\n not alp");
 }
}
```

→ WAP, take one student's 3 marks from KB. Check is passed or not.

```
pf
 ~~int m1, m2, m3;~~
 {
 int p, c, m;
 pf ("enter three marks\n");
 sf ("%d %d %d", &p, &c, &m);
```

Scanned by CamScanner

```
if (p < 30 || c < 30 || m < 30)
  {
   pf ("fail\n");
  }
else
  {
   pf ("pass\n");
  }
}
```

Else if ladder

```
stmt 0;
if (condition1)
stmt 1;
else if (cond 2)
stmt 2;
else if (cond 3)
stmt 3;
else
stmt 4;
stmt 5;
```

$C_1, C_2, C_3 \rightarrow$ True $\rightarrow$ stmt 0, stmt 1, stmt 5

$C_1 \rightarrow F, C_2 \rightarrow T, C_3 \rightarrow T \rightarrow$ stmt 0, stmt 2, stmt 3

$C_1, C_2, C_3 \rightarrow$ False $\rightarrow$ stmt 0, 4, 5

→ when a condition is true, other isn't checked.

Scanned by CamScanner

→ WAP. Take a no from KB. check no is 0, +ve, -ve using else if ladder.

```
{
 int no;
 Pf("enter a no\n");
 sf("%d", &no);
 if (no==0)                    // if (n=0).
 ~~pf("no is 0\n")~~            enter a no
  pf("zero\n");                10
 else if (no>0)                 O/P -> -ve
  pf("+ve\n");
 else
  pf("-ve\n");
}
```

→ WAP. Take a character from KB. check character is Capital alphabet, small alphabet, digit, space or special symbol.

```
{
 ~~int~~ char ch;
  pf("enter a character\n");
  ~~sf("%c", &ch);~~
  ch = getche();
```

Scanned by CamScanner

```
~~if ( ch >= 'A' && ch <= 'Z')~~
if (ch >= 'A' && ch <= 'Z')
pf("\n Ca alp");
else if (ch >= 'a' && ch <= 'z')
pf("\n sm alp");
else if (ch >= '0' && ch <= '9')
pf("\n digit");
else if (ch == ' ')
pf("\n space");
else
pf("\n sy");
}
```

→ WAP. Take 3 nos from KB. Print the smallest no using else if ladder.

```
{
int a, b, c;
pf("enter three nos\n");
sf("%d %d %d", &a, &b, &c);
if (a < b && a < c)
    pf("\n a small");
else if (b < c)
    pf("\n b small");
else
    pf("\n c small");
}
```

Scanned by CamScanner

→ WAP. Take 2 nos from KB. Enter your choice
1. addition
2. Substraction
3. multiplication.

```
{
int no1, no2, ch;
pf ("enter two nos\n");
sf ("%d %d", &no1, &no2);
pf ("enter your choice\n 1. add \n2. sub\n3. mult) :
sf ("%d", & ch;)
if (ch==1)
        pf ("add=%d\n", no1+no2);
else if (ch==2);
      pf ("sub = %d \n", no1-no2);
else if (ch==3);
     pf ("mult=%d \n", no1 * no2);
else
  pf ("\n invalid choice");
}
```

Scanned by CamScanner

```
-> {
    int no1, no2;
    char ch;
    pf("enter two nos\n");
    sf("%d%d", &no1, &no2);
    pf("\nenter your choice + - * \n");
    ch = getche();
     if (ch==='+').
        pf("add=%d\n", no1+no2);
     else if (ch==='-'),
      pf("sub=%d\n", no1-no2);
      else if (ch==='*'),
       pf("mult= %d\n", no1*no2);
     else
        pf("\n invalid choice");

    }
```

-> WAP. enter your choice. take input from KB.

1. Area of rectangle
2. " " square
3. " " circle

```
{
    float l, b, s, r;
    int ch;
    pf("\nenter your choice\n1. area of rectangle
      \n2. area of square\n3. area of circle");
```

Scanned by CamScanner

```
sf("%d", &ch);
if (ch==1)
  {
   pf("enter rectangle l and b\n");
   sf("%f%f", &l, &b);
   pf("area of rectangle =%.2f\n",
                          , l*b);
  }
else if (ch==2)
   {
    pf("enter square s);
    sf("%f", &s);
    pf("area of square =%.2f\n", s*s);
   }
else if (ch==3)
   {
    pf("enter ~~squ~~ circle r);
    sf("%f", &r);
    pf("area of circle =%.2f\n", 3.14*r*r);
   }
else
   {
    pf("invalid choice\n");
   }
}
```

Scanned by CamScanner

Switch case

-> switch is a keyword.

Syntax

```
switch(condition)
{
case lebel 1: stmt; break;
:
:
case lebeln : stmt; break;
default : stmt;
}
```

optional

break; -> optional

-> switch, case, default -> keyword

-> Switch condition must be integer type. (char, int, enum).

~~-> It~~ It may variable or const.

-> Switch condition can't be float and double type.

```
switch(3)
{
}
```

✓

```
switch(3.4)
{
}
```

✗ error

-> Switch case lebel must be integer const. (char, int, enum)

-> Case lebel can't be variable

-> Case lebel can't be float & double const.

Scanned by CamScanner

```
Switch (3)
{
Case 3: pf("okok");
}
```

Valid

```
switch(3)
{
Case 3.0: pf("hi");
}
```

invalid → float

```
Switch(3)
{
case a: pf("byebye");
}
```

→ Variable invalid

→ Case label may be random order

→ What is the value generated in condition, match which case ~~that pos~~, program is executed from that point.

```
switch(3)
{
case 2: pf("B");
Case 3: pf("C");
Case 1: pf("A");
}
```

O/P → C A

Scanned by CamScanner

break

-> break is a keyword.

-> It is used in switch or loop.

-> break prevents the next case and execute outside the switch block.

-> It is optional in switchcase.

```
switch(2)
{
 case 2: pf("B"); ~~brea~~        -> break
 case 3: pf("C"); break
 case 1: pf("A");
}
```

O/P -> B

O/P -> BC

-> Case label is always unique. Two case labels can't be same.

```
switch(2)
{
 case 2: pf("B");
 case 2: pf("C");
}
```

X error

Duplicate case

Scanned by CamScanner

```
Switch (2)
{
 Case 97: ...
 Case 'a': ...
}
```

X error

~~Ashi~~ → Duplicate case

→ If there is no case, there is no error

```
{
 Switch (2)
  {
  }
}
```

No output

→ There is no case match, there is no error

```
{
 switch (2)
 {
  Case 3: pf( );
 }
}
```

No output

default

→ It is a keyword.

→ It is used only in switch. case.

→ If there is no ~~face~~ case match default will be executed.

Scanned by CamScanner

```
{
  switch (3)
  {
   case 2: pf("B");
   default: pf("no match");
  }
}
```

O/P -> no match

-> In default there is no lebel. So we write on default.

-> It can be used at any place inside the switch blak. But a good programmer write at last.

-> default is optional.

```
-> {
   switch (3)
   {
    default: pf("no match");
    case 2: pf("B");
   }
}
```

O/P -> no match B

```
-> {
   switch (3)
   {
    default: pf("no match"); break;
    case 2: pf("B");
   }
}
```

O/P -> no match

Scanned by CamScanner

continue

→ continue is a keyword.

→ continue is not used only in switch case.

→ In switch case inside the loop, we can use continue.

```
{
  switch(3)
  {
    case 2 : pf("B"); continue;
  }
}
```

→ switch(0) must be evaluated.

if(0) X

while(0) X

→

```
{
  switch(0)
  {
    case 0 : pf ("zero");
  }
}
```

O/P → zero

→ ~~WAP~~ switch condition, label must be integer type expression.

```
{
  Switch(2*3)
  {
    case 1+3+2 : pf ("A"); break;
    case 2*3 : pf ("B"); break;
  }
}
```

X Duplicate case

Scanned by CamScanner

```
{ switch (2*3)
  {
   case 1+3+2 : pf ("A"); break;
   case 2*3 : pf ("B"); break;
  }
}
```

O/P -> A

-> WAP, take a no from KB. check no is even or odd using switch case.

```
{
 int no;
 pf ("enter a no\n");
 sf ("%d", &no);
 switch (no % 2 == 0)
 {
  case 1 : pf ("even"); break;
  ~~case 0~~
  default : pf ("odd"); break;
 }
}
```

->

```
{
 int no1, no2, ch;
 pf ("enter two nos\n");
 sf ("%d%d", &no1, &no2);
 pf ("enter you choice
    \n 1. add \n 2. sub \n 3. multi");
```

Scanned by CamScanner

```
sf ("%d", &ch);
switch(ch)
{
case 1: pf ("add = %d\n", no1+no2); break;
case 2: pf ("sub = %d\n", no1-no2); break;
case 3: pf ("mult = %d\n", no1*no2); break;
default: pf("\n invalid choice");
}
```

→ WAP. Enter any month no and print the month.

```
{
int month;
Pf ("enter a month\n");
sf ("%d", &month);
switch(month)
{
case 3: case 4: case 5: case 6: pf("summer");
case 7: case 8: case 9: case 10: pf("rainy");
case 11: case 12: case 1: cas 2: pf("winter");
default: pf("\n invalid ~~choice~~");
}
```

Scanned by CamScanner

→ WAP. take an alphabet from KB. Check vowel or not.

```
{
 char ch;
 pf("enter a ch\n");
 sf("%c", &ch);
 switch(ch)
 case 'a': case 'A': case 'e': case 'E' - - - -
   pf("vowel");
 default pf("consonant");
}
```

→

```
{
 switch(2)
 case 2: pf("hi");
}
```

O/P → hi

→ if there is single statement in switch case { } isn't required.

→

```
switch(2)
 case 2: ---
 case 3: ---
```

X case isn't within the switch

Scanned by CamScanner

```
-> {
   switch(2)
   case 2: pf("hi"); break;
   }
```

X break statment isn't within switch

-> Any stmt inside the switch must be under a case.

```
{
switch(2)
{
pf("hi");
}
}
```

-> No output
No error

```
-> {
   switch(2)
   {
   pf("hi");
   case 2: pf("ok ok");
   }
   }
```

O/p -> ok ok

Scanned by CamScanner

Nested switch

→ Switch is in switch ~~ch~~, that switch must have ~~case~~ be a case.

```
void main()
{
 switch(2)
 {
   case 1: pf("A");break;
   case 2: switch(2)
   {
     case 1: pf("B"); break;
     case 2: pf("C"); break;
     default: prf("D")
   }
   default: Pf("E");
 }
}
```

O/P → CE

→ WAP. Take a no from KB. Check no is odd, even, or 0. using nested switch case.

```
Void main()
{ int no;
  pf("enter a no\n");
  sf("%d", &no);
  switch(no)
  {
   case 0: pf("zero");break;
   default: switch(no%2==0)
```

Scanned by CamScanner

```
}
case 1: pf ("even no"); break;
default: pf ("odd no");
}
}
```

Looping

If Some statement is repeated continuously then we go for loop.

The repeated stmt writing once inside the loop bodypart.

There are three types of loop.

1 -> While loop

2 -> Dowhile loop

3 -> for loop

In every loop there are three sections.

1) Initialisation section (start value)

2) Conditional section (stop value)

3) Increment decrement section (step value/updatable value)

Scanned by CamScanner

While loop

syntax

```
init; ——> optional
while (condition) —> Compulsory
{
    //repeated statement is written once
    inc/dec; —> optional
}
```

→ WAP, print hello msg five times using while loop.

```
void main()
{
    int i=1;
    while (i<=5)
    {
        pf ("hello\n");
        i++;
    }
}
```

True

False

```
i=5
(i>=1)
{
    i--
}
```

O/P

hello
hello
hello
hello
hello

→ WAP, print 1 to 10 using while loop.

```
void main()
{
    int i=1;
    while (i<=10)
    {
        pf ("%d\n", i);
        i++;
    }
```

Scanned by CamScanner

→ WAP, print 10 to 1 using while loop

```
{
 int i=10;
 while (i >= 1)
 {
  pf ("%d\n", i);
  i--
 }
}
```

<u>break & continue used in while loop</u>

```
while (condition)
{
 break;
}
```

```
while (condition)
{
 continue;
}
```

```
void main()
{
    int i=1;
    pf ("A\n");
    while (i<=4)
    {
     pf ("B\n");
     i++;
     break; // continue
     pf ("C\n");
    }
    pf("%d\n", i);
}
```

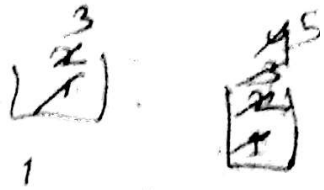
O/P-1 →

A
B
2

O/P - 2 →

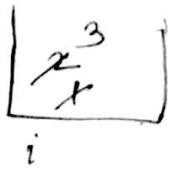
A
B
B
B
B

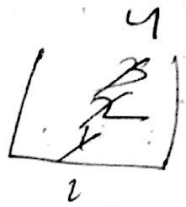
```
{
int i=1;
pf ("A\n");
while (i<=4)
{
  pf ("B\n");
  i++;
  if (i>=3)
  {break; } //continue;
  pf ("C\n");
}
  pf ("%d\n", i);
}
```



break	continue
O/P	O/P
A	A
B	B
C	C
B	B
3	B
	B
	5

```
{
int i=1;
pf ("A\n");
while (i<4)
{ pf ("B\n");
  i++;
  if (i==3)
  {break;} // continue;
  pf ("C\n");
}
pf ("%d\n", i);
}
```





break	Continue
O/P	O/P
A	A
B	B
C	C
B	B
3	B
	C
	4

Scanned by CamScanner

Do while loop

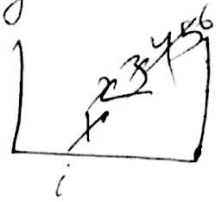
syntax

```
int;
do
{
  //repeat stmt
  inc/dec;
} while (condition);
```

True

False

-> WAP, print hello 5 times using do while loop



```
-> void main()
   {
->   int i=1;
     do
     {
->     pf("hello\n");
       i++;
     } while (i<=5);
   }
```

O/P

hello
hello
hello
hello
hello

-> WAP. print 1 to 7 using do while loop

```
#include <stdio.h>
void main()
{
  int i=1;
  do
  {
    pf("%d\n", i);
    i++;
```

Scanned by CamScanner

```
} while (i<-1);
}
```

→ WAP. Print 10 to 1 using do while loop

```
void main()
{
 int i=10;
 do
 {
   pf ("%d\n", i);
   i--;
 } while (i>=1);
}
```

While

```
{
pf ("A");
while (0)
{
pf ("B");
}
pf ("C");
}
```

O/P → AC

do while

```
{
 pf ("A");
 do
  {
   pf ("B");
  } while (0);
  pf ("C")
}
```

O/P → ABC

This is called at least once.

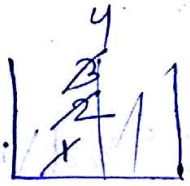
Scanned by CamScanner

: break & continue used in do while loop

```
do
{
  break;
} while (condition)
```

```
do
{
  continue;
} while (condition);
```

```
void main()
{
  int i=1;
  pf ("A\n");
  do
  {
    pf ("B\n");
    ~~if(i>3)~~
    ~~} while (1)~~
    i++;
    if (i>3)
    break; //continue;
    ~~}while~~
    printf ("c\n");
  } while (i<5);
    pf ("%d\n",i);
}
```



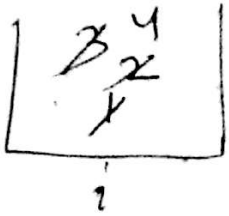
i

break
A
B
C
B
C
B

break	Continue
A	A
B	B
C	
B	
C	
B	

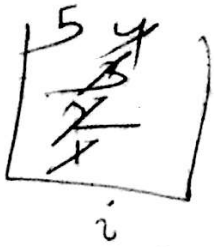
Scanned by CamScanner

break



A
B
C
B
C
B
4

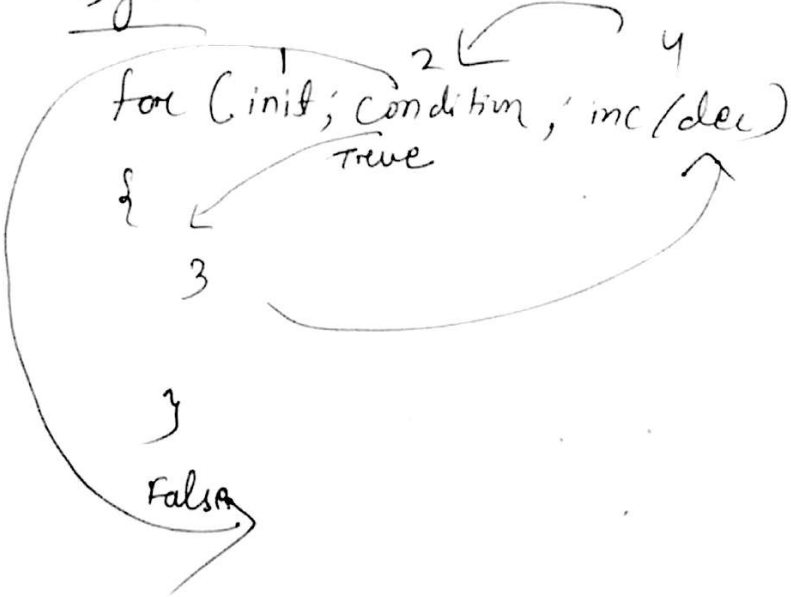
Continue



A
B
C
B
C
B
B
5

For loops

syntax



```
{
 int i;
 for (i=1; i<=5; i++)
 {
   pf ("hello\n");
 }
}
```

O/P → hello
hello
hello
hello
hello

Scanned by CamScanner

→ WAP. print 1 to 10 using for loop.

```
{
 int i;
 for (i=1; i<=10; i++).
  {
   pf("%d\n", i);
  }
}
```

→ 10 to 1

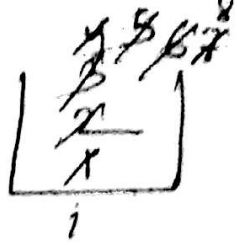
```
{
 int i;
 for (i=10; i>=1; i--).
  {
   pf("%d\n", i);
  }
}
```

<u>break</u> & <u>continue</u> <u>using</u> <u>for</u> <u>loop</u>

```
for (init; condition; inc/dec)
{
  break; // continue;
}
```

Scanned by CamScanner

```
void main()
{
 int i;
 for (i=1; i<=7; i++)
 {
  pf ("A");
  if (i>=4)
  break; //continue;
  pf ("%d\n", i);
 }
 pf("%d\n",i);
}
```

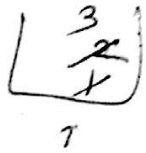


break	Continue
A1	A1
A2	A2
A3	A3
A4	AAAA8

```
void main()
{
 int i=1;
 for (pf ("A"); pf ("B"); pf ("C"))
 {
  pf ("D");
  i++;
  if (i==3)
   break;
  pf ("E");
 }
 pf ("F");
}
```

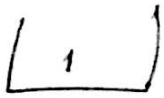
O/P

ABDECBDF



Scanned by CamScanner

```
void main()
{
 int i;
 for (i=1; i<=5, i+2)
 pf("%d\n", i);
}
```

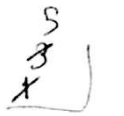


O/P

1
1
1
1
1
1
1
∞

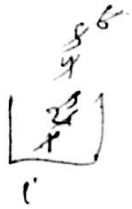
```
for (i=1, i<=5, i=i+2)
pf("%d\n", i);
```

O/P



~~1 3 5~~
1
3
5

```
void main()
{
 int i;
 for (i=1, i<=5; i++)
 pf("%d\n", i);
}
```



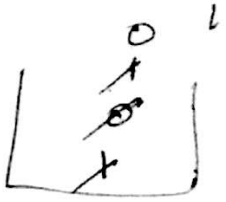
O/P -> 6.

```
void main()
{
int i;
for (i=1; i<=10; i++)
{if (i%2==0)
 pf ("%d\n", i=i+3);
```

Scanned by CamScanner

```
else
pf("%d\n", i=i-1);
pf("%d\n", i);
}
```

//pf("%d\n", i=i+1)

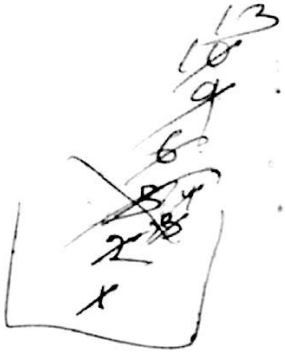


O/P else (i-1)

0
infinte loop
5
2
7

O/P else (i+1)

2
4
6
8
10
11
13



Lebel

Valid identifiere with colon (:)

```
void main()
{
  ab:
  pf("hello");
}
```

O/P -> hello

Scanned by CamScanner

goto is a keyword. It is used to transfer the control one location to another location.

syntax

goto labelname;

> It is used in two ways.

1. forward goto
2. backward goto

```
{
  pf("A")
  goto puri;
  pf("B");
}
```

X error
label "puri" used is not defined.

forward

```
void main()
{
  pf("A");
  goto puri;
  pf("B");
  pf("C");
  puri:
  pf("D");
}
```

O/P -> A D

Scanned by CamScanner

backward

```
{
 pf("A");
puri:
 pf("B");
 pf("c");
 goto puri;
 pf("D");
}
```

O/P

ABCBCBC - - - -

~~backward~~ goto with condition

Conditional goto

if goto is a condition, it is called conditional goto.

backward goto with condition

What are the programs solved in loop, those programs can be solved by backward goto with condition.

```
{
 int i=1;
 pf("A");
 puri:
 pf("B");
 pf("c");
 i++;
 if(i<4)
 goto puri;
 pf("D");
}
```

O/P

ABCBCBCD

Scanned by CamScanner

→ WAP. Print 1 to 10 using goto.

```
{
 int i=1;
 no:
 pf("%d \n",i);
 i++;
 if(i<=10)
  goto no;
}
```

→ WAP to print

Hello - 5 times

Hi - 4 times

byebye - 7 times using 3 variables.

→

```
{
 int i,j,k;
 i=1;
 while(i<=5)
 {pf("Hello\n");
  i++;
 }
 j=1;
 while (j<=4)
  {
   pf("Hi\n");
   j++
  }
 k=1;
 while(k<=7);
  ~~k++~~ pf("byebye\n");
  k++;
```

Using one variable

```
int i;
i=1;
while (i<=5)
{pf("Hello\n");
i++;
}
i=1;
while (i<=4).
{
 pf("Hi\n");
 i++
}

i=1
while (i<=7);
{pf("byebye\n");
i++
}
```

Scanned by CamScanner

→ Using one variable

```
{
int i;
i = 1;
```

→ Print n natural nos.

```
void main()
{
int i=1, n;
pf("enter a no\n");
sf("%d\n", &n);
while (i<=n)
{
pf("%d\n", n);
i++;
}
```

→ To print

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

for

```
void main()
{
int i,j;
for(i=1; i<=3; i++)
{
for (j=1, j<=4; j++)
{
pf("*\t");
}
pf("\n");
}
```

while

```
void main()
{
int i,j;
i=1
while (i<=3)
{
~~while(j~~
j=1;
while(j<=4)
{
pf("*\t");
j++;
}
pf("\n");
i++;
}
}
```

do while

```
void main()
int i,j;
do
{ i=1
j=1;
do
{
pf("*\t");
j++;
} while (j<=4);
pf("\n");
i++;
} while (i<=3);
}
```

Scanned by CamScanner

```
{
int i,j, r,c;
pf("enter r and c\n");
sf("%d%d", &r, &c)
for (i=1 ; i<=r ; i++)
  {
    for (j=1 ; j<=c ; j++)
    {
      pf("*\t");
    }
    pf("\n");
  }
}
```

~~brea~~ break & continue used in nested for loop

```
for(init ;condition; inc/dec)
  {
    for(init ;condition;inc/dec)
    {
      break ;//continue
    }
    break ;//continue
  }
```

Scanned by CamScanner

```
> {
  int i=1, j=1;
  for (pf ("A"); pf ("B"); pf ("C"))
   { i++; j++
     pf ("D");
     for (pf ("E"); pf ("F"); pf ("G"));
     }
     if (i==2 && j<=2)
       ~~pf ("*")~~
       break;
       pf ("i");
     }
     if (i==3)
       break;
     pf ("j");
     ~~pf~~ i++;
          j++;
          pf ("\n");
     }
  }
```

-) ~~A B D E F I G F~~

A B D E F i G F

```
void main()
{
  int i=1, j=1;
  for (i=1; i<=3; i++)
  {
    for (j=1; j<=3; j++)
    {
      if (i==j)
        break; // continue
        pf("%d %d\n", i, j);
    }
    pf("\n");
  }
}
```

O/P → break

2 1

3 1

3 2

Continue

1 2
1 3

2 1
2 3

3 1
3 2

```
{ int i, j;
  for (i=1; i<=3; i++)
  {
    for (j=1; j<=4; j++)
    {
      pf("*\t");
    }
    pf("\n");
  }
}
```

O/P ⇒

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

pf("%d\t", i);	pf("%d\t", j);
O/P → 1 1 1 1 2 2 2 2 3 3 3 3	1 2 3 4 1 2 3 4 1 2 3 4

Scanned by CamScanner

(i--, j--)

```
-> {
   int i,j;
   for (i=3; i>=1; i--)
   {
    for (j=4; j>=1; j--)
    {
     pf("%d\t");
    }
    pf("\n");
   }
}
```

① ~~i++~~ i

3 3 3 3
2 2 2 2
1 1 1 1

② ~~j--~~ j

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

(i++, j--)

```
-> {
   int i,j;
   for (i=1; i<=3; i++)
   {
    for (j=4; j>=1; j--)
    {
     pf("%d\t");
    }
    pf("\n");
   }
}
```

① i

1 1 1 1
2 2 2 2
3 3 3 3

j

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

-> i--, j++

```
{
 int i, j;
 for (i=3; i>=1; i--)
 {
  for (j=1; j<=4; j++)
  {
   pf("%d\t");
  }
  pf("\n");
 }
}
```

i

3 3 3 3
2 2 2 2
1 1 1 1

j

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

Scanned by CamScanner

```
void main()
{
  int i,j;
  for(i=1;i<=4;i++)
  {
    for(j=1;j<=i;j++)
    {
      pf("*\t");
    }
    pf("\n");
  }
}
```

O/P ->

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

```
pf("%d\t",i);
```

O/P ->

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

```
pf("%d\t",j);
```

O/P ->

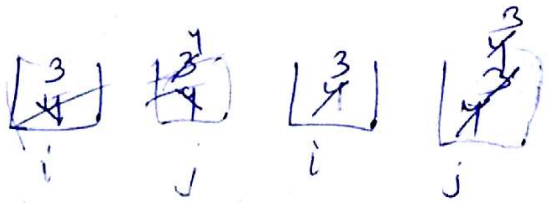
```
1
2 2
3 3 3
4 4 4 4
```

-> (i--, j--)

```
void main()
{
  int i,j;
  for(i=4;i>=1;i--)
  {
    for(j=4;j>=i;j--)
    {
      pf("*\t");
    }
    pf("\n");
  }
}
```

O/P ->

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




```
pf("%d\t",i)
4
3 3
2 2 2
1 1 1 1
```

```
pf("%d\t",j)
4
4 3
4 3 2
4 3 2 1
```

Scanned by CamScanner

→ (i++, j--)

```
void main()
{
 int i, j;
 for (i=1; i<=4; i++)
  {
   for (j=i; j>=1; j--)
   {
    pf("*\t");
   }
   pf("\n");
  }
}
```

O/p →

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

pf("%d\t", i)

O/p →

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

pf("%d\t", j)

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

→ (i--, j++)

```
void main()
{
 int i, j;
 for (i=4; i>=1; i--)
  {
   for (j=1; j<=4; j++)
   {
    pf("*\t");
   }
   pf("\n");
  }
}
```

O/p →

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

j

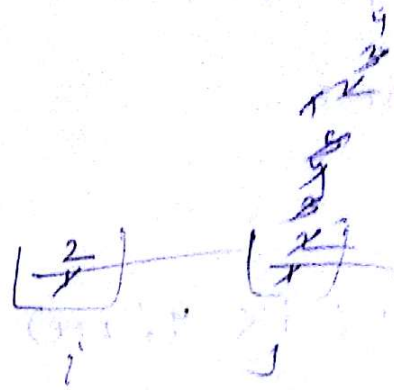
```
4
3 4
2 2 3
1 2 3 4
```

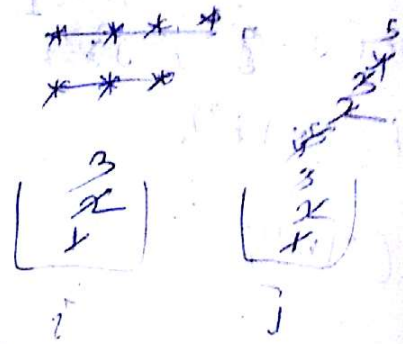
Scanned by CamScanner

-) * * * *
* * *
* *

->(i+1, j++)

```
void main()
{
int i, j;
for (i=1; i <= 4; i++)
{
 for(j=i; j <= 4; j++)
 {
   pf("*\t");
 }
 pf("\n");
}
}
```





* * * *
* * *
* *
*

-> pf("%d\t", j)

1 2 3 4
2 3 4
3 4
4

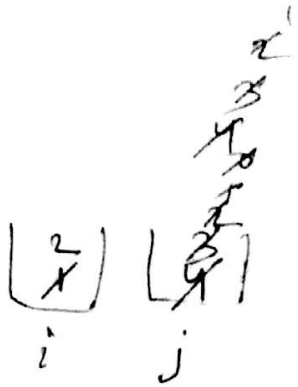
-> (i--, j--)

```
}
int i, j;
for (i=4; i>=1; i--)
{
 for (j=i; j>=1; j--)
 {
   pf("*\t");
 }
 pf("\n");
}
}
```

Scanned by CamScanner

→ (i++ , j--)

```
void main()
{
 int i,j;
 for( i=1; i<=4; i++)
 {
  for( j=4; j>=i; j--)
  {
   pf("*\t");
  }
  pf("\n");
 }
}
```

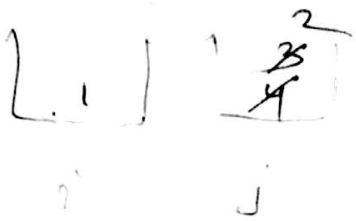



```
* * * *
* * *
* *
*
```

~~→ pf~~ → pf("%d\n", i)

~~→ pf("~~

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



→ pf("%d\n", j)

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

Scanned by CamScanner

→ (i--, j++)

```
for(i=4; i>=1; i--)
{
  for(i=1; j<=i; j++)
  pf
}
pf
```

→

```
void main()
{
  int i, j;
  for(i=65; i<=67; i++)
  {
    for(j=65; j<=68; j++)
    {
      pf("%c\t", j);
    }
    pf("\n");
  }
}
```

A A A A
B B B B
C C C C

O/P → A B C D
A B C D
A B C D

① A
A B
A B C
A B C D

② 1) C B A
C B A
B A
A

```
void main()
{
 int i, j;
 for(i=65; i<=68, i++)
 {
  for (j=65; j<=i; j++)
  {
   pf("%c\t", j);
  }
  pf("\n");
 }
}
```

O/P →

```
A
A B
A B C
A B C D
```

```
void main()
{
 int i, j;
 for (i=68; i>=65; i--)
 {
  for (j=i; j>=65; j--)
  {
   pf("%c\t", j);
  }
  pf("\n");
 }
}
```

Scanned by CamScanner

```
for(i=65; i<=68; i++)
for (j=i; j<=68; j++)
```

~~AB~~

```
A B C D
A B C
A B
A
```

—— x ——

```
{
int i, j;
char ch = 'A';
for (i=1; i<=4; i++)
 {
  ch = 'A';
  for (j=1; j<=i; j++)
  {
   pf ("%c\t", ch);
   ch++;
  }
  pf ("\n");
 }
}
```

O/P ->

```
A
A B
A B C
A B C D
```

Scanned by CamScanner

```
{int i,j;
 char ch='D';
 for (i=1; i<=4; i++)
 {
  ch='D';
  for (j=1; j<=i; j++)
  {
   pf ("%c\t", ch);
   ch--;
  }
  pf ("\n");
 }
}
```

O/P ->

```
D
D C
D C B
D C B A
```

```
-> {
 int i,j;
 char ch='A';
 for (i=4; i>=1; i--)
 {
  for (j=1; j<=i; j++)
  {
   pf ("%c\t", ch);
   ch++;
  }
  pf ("\n");
 }
}
```

O/P

```
A B C D
E F G
H I
J
```

Scanned by CamScanner

```
{
int i, j;
Int ctr = 0;
for (i=4; i>=1; i--)
{
  for (j=1; j<=i; j++)
  {
   pf ("%d\t", ctr);
   ctr++
  }
}
}
```

O/P

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

```
{
int i, j;
char ch='A', ch1;
for (i=4; i>=1; i--)
{
  ch1=ch;
  for (j=1; j<=i; j++)
  {
   pf ("%c\t", ch1);
   ch1++;
  }
  ch++;
  pf ("\n");
}
```

O/P -> AB C D
B C D
C D
D

```
-> #include <stdio.h>
   #include <string.h>
   void main()
   {
   int i, j;
   char str[20];
   pf("enter a string\n");
   gets(str);
   for (i=0; i<strlen(str); i++)
   {
     for (j=0; j<=i; j++)
     {
       pf("%c \t", str[j]);
     }
     pf("\n");
   }
   }
```

O/P

```
L
L a
L a k
L a k s
L a k s h
L a k s h y
L a k s h y a
```

```
-> #include <stdio.h>
   void main()
   {
   int i, j;
   for (i=1, i<=4; i++)
   {
   for (j=1; j<=i; j++)
   { pf ("%d\t", i%2); // pf ("%d\t", !(i%2));
   }
   pf ("\n");
   }
   }
```

O/P

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

O/P

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

Scanned by CamScanner

~~(i+j)%2~~ →

```
{
  int i, j;
  for (i=1; i<=4; i++)
  {
    for (j=1; j<=i; j++)
    {
      pf ("%d\t", (i+j)%2);
    }
    pf ("\n");
  }
}
```

O/P

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

//!(i+j)%2 →

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

```
{
  int i, j;
  for (i=1; i<=4; i++)
  {
    for (j=i-1; j>=1; j--)
    {
      pf ("  ");
    }
    for (j=1; j<=4; j++)
    {
      pf ("*");
    }
    pf ("\n");
  }
}
```

j=1; j<=i; j++

O/P

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

→ pf ("%d\t", j);

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

Scanned by CamScanner

```
{
int i,j;
for (i=1; i<=4; i++);
{
 for (j= i ; j<=4 ; j++);
 {
   pf ("  ");
 }
 for (j=1 ; j<=i ; j++)
  {
   pf ("*");
  }
  pf ("\n");

 }
```

O/P

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

```
-> {
 int i,j;
 for (i=1; i<=4; i++)
 {
    for (j= i ; j< 4 ; j++)
    { pf ("  ");
    }
    for (j=1; j<=i ; j++)
     {pf ("%d",j);
     }
    for (j=i-1; j>=1 ; j--)
     {pf ("%d",j);
     pf ("\n"); }}
```

O/P

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

Scanned by CamScanner

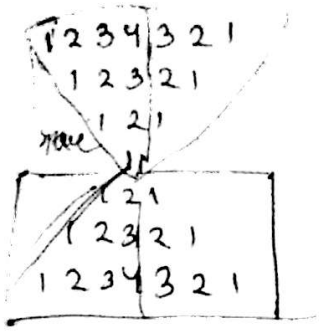
Copy

```
for (i=1; i<=3; i++)
{
 for (j=i; j>=1; j--)
   pf (" ");
 for (j=1; j<=4-i; j++)
 {
   pf ("%d",j);
 }
 for (j=3-i; j>=1; j--)
   pf ("%d",j);
   pf ("\n");
}
```

op

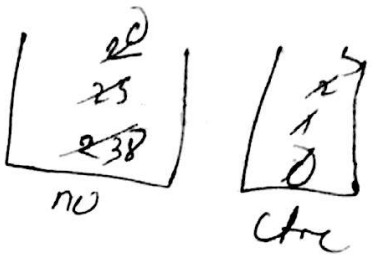
```
    1
   1 2 1
  1 2 3 2 1
 1 2 3 4 3 2 1
  1 2 3 2 1
   1 2 1
    1
```

Scanned by CamScanner



→ WAP. take a no from KB. Count the no of digit.

```
void main()
{
  int no, ctr=0;
  pf("enter a no\n");
  sf("%d", &no);
  while (no!=0)
  {
    ctr++;
    no=no/10;
  }
  pf("no of digit = %d\n", ctr);
}
```




```
do
{
  ctr++;
  no=no/10;
} while (no!=0
```

```
for(; no!=0;)
{
  ctr++;
  no=no/10;
}
```

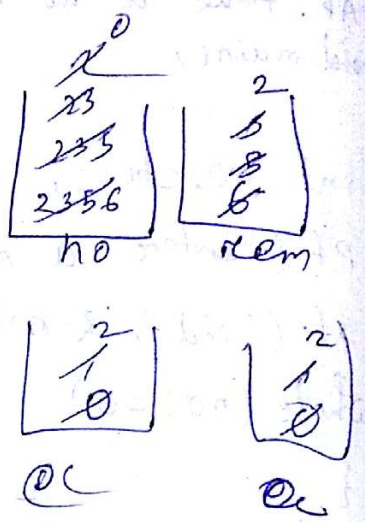
→ WAP. Take a no from KB. Display all the digits in inverse order.

```
{
  int no;
  pf("enter a no\n");
  sf("%d", &no);
  while (no!=0)
  {
    pf("%d\n", no%10);
    no=no/10;
  }
```

Scanned by CamScanner

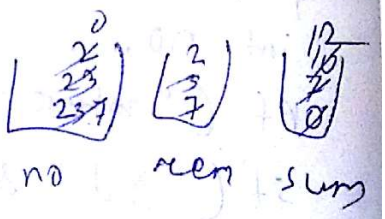
→ WAP. Take a no from KB. Count no of even and odd digit.

```
{
    int no, rem, oc = 0, ec = 0;
    pf("enter a no\n");
    sf("%d", & no);
    while (no != 0)
    {
      rem = no % 10;
      if (rem % 2 == 0)
       ec++;
      else
       oc++;
      no = no / 10;
    }
    pf("no of even digit = %d\n", ec);
    pf("no of odd digit = %d\n", oc);
}
```



→ WAP. Take a no from KB. find the sum of digit.

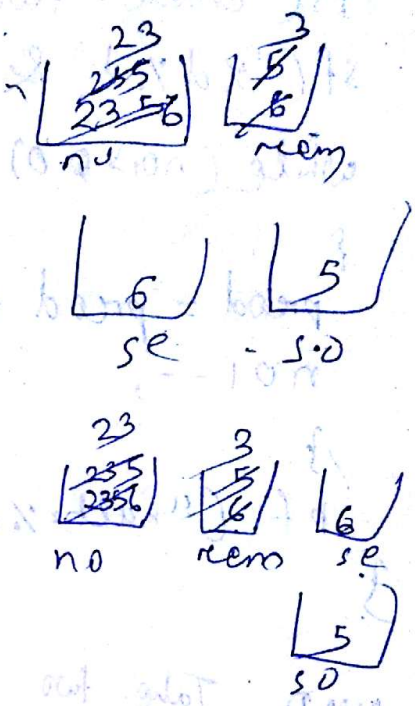
```
{
int no, rem, sum = 0;
pf("enter a no\n");
sf("%d", & no);
while (no != 0)
{ rem = no % 10;
  sum = sum + rem;
  no = no / 10; }
pf("sum of digit = %d\n", sum) }
```



Scanned by CamScanner

→ WAP. Take a no from KB. Find the sum of even & odd digits.

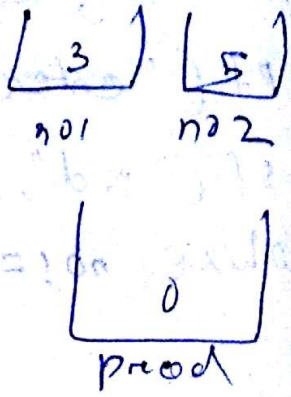
```
{
  int no, rem, se = 0, so = 0;
  pf("enter a no\n");
  sf("%d", &no);
  while(no!=0)
  {
    rem = no % 10
    if (rem % 2 == 0)
      ~~se = se + rem~~
    {
      ~~pf( se = rem~~ ;
        se = se + rem;
      } { se = se + rem;
    }
    else {~~so = rem;~~
        so = so + rem;
    }
    no = no/10
  }
  pf("sum of even digit = %d", se);
  pf("sum of odd digit = %d", so);
}
```



BARSHA

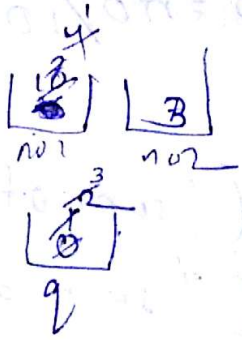
→ WAP. Take 2 nos from KB. multiply without any multiplication operator.

```
{
int no1, no2, prod = 0;
pf("enter two nos\n");
sf("%d %d", &no1, &no2);
while (no1 > 0)
{
 prod = prod + no2;
 no1--;
}
pf("mult = %d\n", prod);
}
```



→ WAP Take two nos from KB. Find the ~~cocent~~ quoscent & remainder without using division & modulo operation.

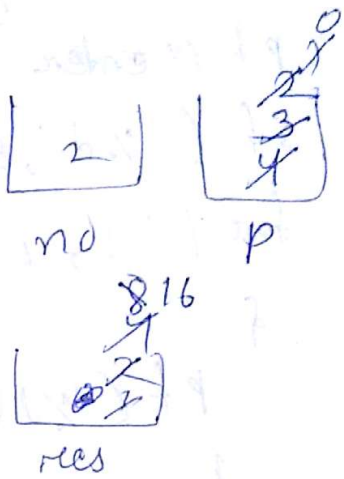
```
{
int no1, no2, q=0, r;
pf("enter two nos\n")
sf
while (no1 > no2)
{
 q++;
 no1 = no1 - no2;
}
pf("q = %d\n", q);
pf("rem = %d\n", no1);
}
```



Scanned by CamScanner

→ WAP. Take a no from KB. Find the power without using power funⁿ.

```
{
int no, p, res = 1;
pf ("enter no & p\n");
sf ("%d %d", &no, &p);
while (p!=0)
{
res = res * no;
p--
}
pf ("result = %d\n", res);
}
```



→

```
{
int no, i, p, n result = 1
pf ("enter a no and no of term\n")
sf ("%d%d", &no, &n);
pf ("1\t");
for (i=1; i<n; i++)
{
p = i;
while (p!=0)
{
result = result * no;
p--;
}
pf ("%d\t", result);
result = 1;
}}
```

O/P

1 2 4 8

→ Display n natural no.

```
{
 int i=1, range;
 pf("enter a range\n");
 sf("%d", &range);
 for (i=1; i<=range; i++)
 {
   pf("%d\t", i);
 }
}
```

→ Sum of n natural nos.

$\frac{n(n+1)}{2}$

```
{
 int i=1, range, sum=0;
 pf("enter a range\n");
 sf("%d", &range);
 for (i=1; i<=range; i++)
 {
  sum=sum+i;
 }
 pf("sum=%d\n", sum)
```

i

sum = 1

sum = 2

sum = ~~sum~~(range*(range+1))/2

-) WAP. take n natural no from KB, find the sum of even no & odd no.

```
{
 int i=1, range, esum=0, osum=0;
 pf("enter a range\n");
 sf("%d", &range);
 for(i=1; i<=range; i++)
 {
  if (i%2==0)
     esum = esum+i;
  else
     osum = osum+i;
 }
 pf("esum = %d\n", esum);
 pf("osum = %d\n", osum);
}
```

→ To print like 1+2+3+ .... n = sum of n natural nos

```
{
 int i=1, range, sum=0;
 pf("enter a range\n");
 sf("%d", &range);
 sum = (range*(range+1))/2;
 for(i=1; i<range; i++)
 {
  pf("%d+", i);
 }
 pf("%d = %d", i, sum);
}
```

O/P

3

1+2+3 = 6

Scanned by CamScanner

$i + 3 + 5 = 9$

```
{
int i = 1, range, sum = 0, ctr = 1;
pf ("enter a range\n");
sf ("%d", & range);
for (i = 1; i < range; i++)
{
  sum = sum + ctr;
  pf ("%d +", ctr);
  ctr = ctr + 2;
}
pf ("%d = %d", ctr, sum + ctr);
}
```

i	range	sum	ctr
~~2~~ 3 ~~1~~	3	~~1~~ 4 ~~0~~	~~3~~ 5 ~~1~~

O/P

$1 + 3 + 5 = 9$

```
{
int i = 1, range, sum = 0, ctr = 1;
pf ("enter a range\n");
sf ("%d", & range);
for (i = 1; i < range; i++)
if (i % 2 = = 0)
{ sum = sum + ctr;
  pf ("%d+", ctr);
}
else
{ sum = sum + ctr;
  pf ("%d-", ctr);
```

Scanned by CamScanner

```
    } ctr = ctr + 2;
  }
  if (i % 2 == 0)
  {
    pf("%d = %d", ctr, sum - ctr);
  }
  else
  {
    pf("%d = %d", ctr, sum + ctr);
  }
}
```

O/P

1 - 3 + 5 - 7 = -4

```
{
  int range, a = 0, b = 1, c;
  Pf("enter a range\n");
  sf("%d", & range);
  Pf("%d\t %d\t", a, b);
  while (range > 2)
  {
    c = a + b;
    pf("%d\t", c);
    a = b;
    b = c;
    range --;
  }
}
```

Scanned by CamScanner

Multiplication table

```
{
 int i, no;
 Pf("enter a no\n");
 sf("%d", &no);
 for (i=1; i<=10; i++)
 {
  pf("%d * %d = %d\n", no, i, no*i);
 }
}
```

```
{
 int i, no, range;
 pf("enter a range\n"); sf("%d", &range);
 for (no=1; no<=range; no++)
 {
  for (i=1; i<=10; i++)
  {
   pf("%d * %d = %d\n", no, i, no*i);
  }
  pf("\n\n");
 }
```

```
{
 int i, j, no;
 Pf("enter a no\n");
 sf("%d", &no);
 for (i=1; i<=3; i++)
 {
  for (j=1; j<=10; j++)
```

Scanned by CamScanner

```
{
  if (i==1)
    Pf ("%d\t", no*j);
  else if (i==2)
    Pf("%d\t",j);
  else
    Pf ("%d\t",no);
 }
 Pf("\n");
 }
}
```

Pf("***\n");

```
>
 {
  int i,j,no;
```

GCD

$$2 \,\big|\, 14, 20$$
$$7, 10$$

$$14 \,\big|\, 20 \,\big|\, 1$$
$$14$$
$$6 \,\big|\, 14 \,\big|\, 2$$
$$12$$
$$2 \,\big|\, 6 \,\big|\, 3$$
$$6$$
$$0$$

GCD

LCM

$$\frac{14 \times 20}{2} = \frac{280}{2} = 140$$

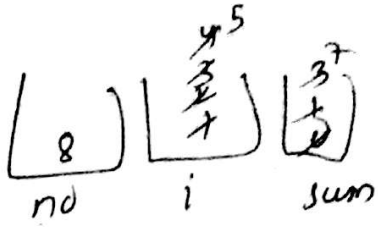
Scanned by CamScanner

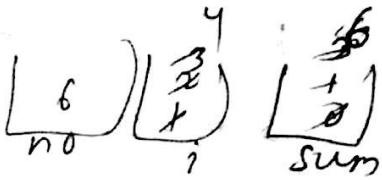
```
{
int no1, no2, n, d, rem;
pf("enter two nos\n");
sf("%d %d", &no1 &no2);
if(no1>no2)
{
  n = no1;
  d = no2;
}
else
{
  n = no2;
  d = no1;
}
rem = n % d
while (rem!=0)
{
  n = d;
  ~~rem~~
  d = rem;
  rem = n % d
}
pf("gcd = %d\n", d);
pf("lcm = %d\n", (no1 * no2)/d);
}
```

Scanned by CamScanner

→ WAP, Take a no from KB. Check it is a perfect no.

```
#include <stdio.h>
void main()
{
 int no, sum=0, i;
  printf("enter a no\n");
   sf("%d", &no);
   for (i=1; i<=no/2; i++)
   {
    if (no % i == 0)
    {
     sum = sum + i;
    }
   }
   if (sum == no)
   pf("%d perfect no\n", no);
   else
   pf("%d not perfect no\n", no);
```





→ WAP. Display all perfect no ~~of~~ upto range.

```
#include <stdio.h>
void main()
{
 int no, sum=0, i, range;
 pf("enter a range\n");
 sf("%d", &range);
```

Scanned by CamScanner

```
for (no=1; no<=range; no++)
{
    //sum=0;
    for (i=1; i<=no/2; i++)
    {
      if (no%i==0)
      {
       sum=sum+i;
       }
    }
    if (sum==no)
    pf("%d perfect no\n", no);
    sum=0;
}
}
```

prime or not

```
# include <stdio.h>
void main()
{
 int no, i, ctr=0;
 pf("enter a no\n");
 sf("%d", &no);
 for (i=2; i<=no/2; i++)
 {
   if (no%i==0)
   {
     ctr++;
     break;
   }
 }
```

36

2<=
3%

```
if (ctr==0)
pf ("%d prime no\n", no);
else
  pf("%d not prime no\n", no);
 }
 }.
```

→ WAP display all prime nos upto range

```
# include <stdio.h>
void main()
{
int no, i, ctr=0, range;
pf ("enter a range\n");
sf ("%d", &range);
for (no=2; no<=range; no++)
 {
  // ctr=0;
  for (i=2; i<=no/2; i++)
  {
   if (no % i==0)
   {
    ctr++;
   break;
   }
  }
  if (ctr==0)
  pf("%d prime no\n", no);
  ctr=0;
 }
```

Scanned by CamScanner

Function

Function has 3 parts.

- function declaration
- function call
- funⁿ deffⁿ

Syntax

→ returntype funⁿ name (arguments);

Void F (void);

F is a funⁿ receive no value return no value

Void add (int, int);

or

void add (int a, int b);

~~F is a~~ add is a funⁿ receive two integer value ~~or~~ return no value.

Syntax

funⁿ definition

returntype funⁿ name (arguments)

Parameter name compulsory

Formal arguments

```
{
  local variable
  //logic
  return;
}
```

Ex:

```
void add (int a, int b)
{
 int s;
 s= a+b;
 pf ("sum = %d\n", s);
 Return;
}
```

Function call

→ whenever function is called fun^n def^n is executed.

→ ~~You~~ we can call a function in 3 ways.

A fun^n having no returnvalue no arguement or with arguement.

```
f(0);
add (10,20);
int a=5, b= 4;
add (a,b);
```

Actual arguement

add (int a, int b) ✗ → shouldn't contain datatype.

Suppose a fun^n returns integer type of value, you can call inside the pf int, for, while.

```
pf ("%d", add (10,20))  <- int add(int a, int b)
                           {
                            return a+b;
                           }
```

Scanned by CamScanner

an

```
int res;
res = add(10,20)
Pf("%d\n", res)
```

-> actual argument -> what are the value passing in function parameter argument such value is known as actual argument. It may be variable, const, expression

-> Formal argument -> what are the values recieved in fun^n def^n parameter, such values are known as formal argument.

-> No of actual argument must be same as no of formal argument.

-> Actual argument & formal argument name is may be same or different.

-> actual argument value is copied into formal argument.

-> Category of fun^n.

No return value no argument - e.g. void

No return value with argument e.g void add (int, int)

return value with no argument e.g int add (void)

return value with argument e.g int add (int, int);

No return value no argument

-> Whenever funⁿ is called, you can't send any value to the funⁿ definition parameter.

-> The funⁿ definition parameter doesn't receive any value.

-> Result value doesn't return to the calling funⁿ.

-> The result output msg must be written inside the funⁿ deffⁿ.

-> In this type of funⁿ you can call directly.

-> WAP. Print a 'hello' msg using funⁿ no return value no argument.

```
# void show
# include <stdio.h>
void main()
{
    void show (void);  ---> funⁿ declaration
    show();            ---> funⁿ call
}
void show()            ---> funⁿ deffⁿ
{
    pf ("hello");
}
```

Scanned by CamScanner

→ WAP. Take 2 nos from KB. add it using fun^n no return value no argument.

```
# include <stdio.h>
void main()
{
void add (int a, int)
Pf ("enter two nos\n");
sf("%d %d", &
```

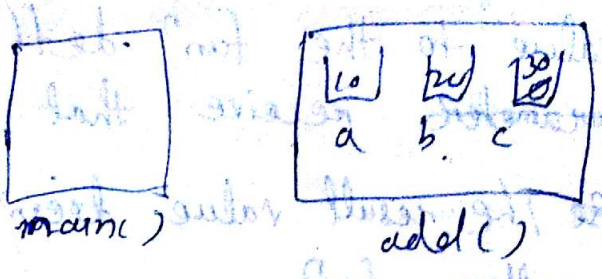
```
# include <stdio.h>
void main()
{
int a, b, s=0;
Pf("enter two nos\n");
sf ("%d %d", & a, &b);
s = a+b;
Pf(" sum = %d\n", s);
}
```

— x —

```
# include <stdio.h>
void main()
{
void add (void);
add();
}
void add ()
{
int a, b, s=0;
pf(" enter two nos\n");
sf (" %d %d", & a, &b;
```

Scanned by CamScanner

```
s = a + b;
pf("sum = %d\n", s);
}
```



→ WAP. Take a no from KB. Check no is even or odd using fun^n no return value no argument.

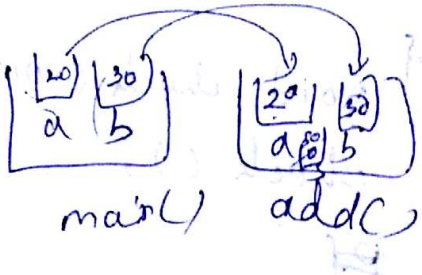
```
#include <stdio.h>
void main()
{
 void check(void);
 check();
 ~~pf~~
}
 void check()
 {
  int no;
  pf("enter a no\n");
  sf("%d", &no);
  if (no % 2 == 0)
  pf("no is even\n");
  else
   pf("no is odd\n");
 }
```

Scanned by CamScanner

No return value with argument

- Whenever funⁿ is called, we can send some value to the funⁿ deffⁿ. A funⁿ deffⁿ ~~parameter~~ receive that values in parameter.
- ~~Be~~ The result value doesn't return to the calling funⁿ.
- The result o/p msg must be written inside the funⁿ deffⁿ.

⇒ WAP. Take two nos from KB. Add it using funⁿ no return value with argument.




```
#include <stdio.h>
void main()
{
void add (int, int);
int a, b;
pf ("enter two nos\n");
sf ("%d %d", &a, &b);
add (a,b);
}
void add (int a, int b)
{
int s=0;
s=a+b;
pf ("sum =%d\n", s);
}
```

fun^n declaration

fun^n call → actual argument

fun^n deffⁿ

→ formal argument

→ WAP. Take a no from KB. Check no. is even or odd using fun^n no return value with argument.

```
#include<stdio.h>
void main( )
{
  void check (int);
  int no;
  pf ("enter a no\n");
  sf ("%d", &no);
  check (no);
}
void check (int no)
{ if (no %2 == 0)
      pf ("no is even\n");
  else
      pf ("no is odd\n");
}
```

2 | 4 | 2 | 0

no
3 % 2 = 1
4 % 2 = 0

→ <u>return value with no argument.</u>

→ Whenever fun^n is called, you can't send any value to the fun^n deff^n.

→ Then Fun^n deff^n doesn't receive any value in parameter.

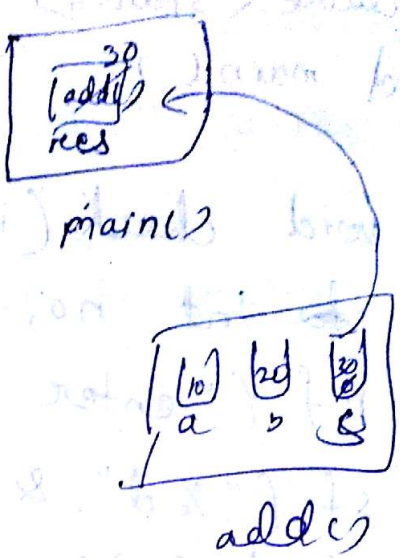
→ The result value return to the calling fun^n.

→ The result o/p msg must be written inside the calling fun^n.

Scanned by CamScanner

→ WAP. Take two nos from KB. Add it using funⁿ return value with no argument.

```
void main()
{
 int add(void);
 int res;
 res = add();
 pf("sum = %d\n", res);
}
int add()
{
 int a, b, s = 0;
 pf("enter two no\n");
 sf("%d %d", &a, &b);
 s = a+b;
 return s;
}
```



→ WAP. Take a no from KB. check no is even or odd using funⁿ return value with no argument

```
# include <stdio.h>
void main()
{
 ~~void~~ int check(void);
 int res;
 res = check();
 pf("
```

```
void main()
{
 int check(void);
 if (check() ==0)
     pf("even no");
 else
     pf("odd no");
}
int check()
{
 int no;
 pf("enter a no\n");
 sf("%d", &no);
 return no%2==0?0:1;
}
```

→ Return value with argument

→ whenever fun^n is called we can pass some value to the fun^n deff^n. Then fun^n deff^n receive that value with parameter.

→ The result value return to the calling fun^n.

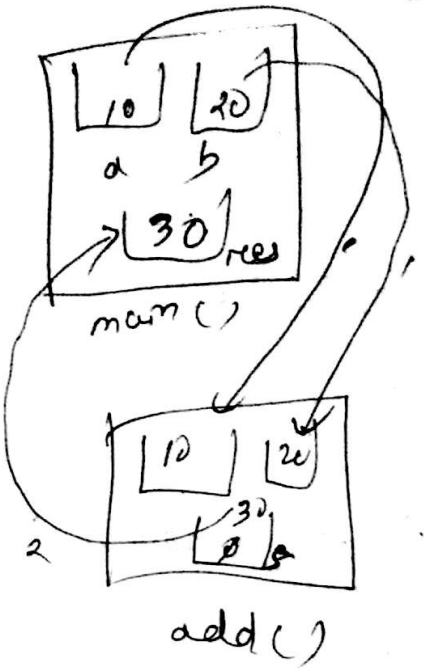
→ The result o/p msg must be written inside the calling fun^n.

WAP. take two nos from KB. add it using fun^n return value with argument.

```
#include <stdio.h>
void main()
{
    int add(int,int);
```

Scanned by CamScanner

```
int a,b,res;
pf("enter two no\n");
sf("%d%d",&a,&b);
res= add(a,b);
pf("sum = %d\n", res);
}
int add(int a,int b)
{
 int s=0;
 s= a+b;
 return s;
}
```



→ WAP. Take a no from KB. Find the factorial with out using fun. except main fun.

```
void main()
{
 int no, f=1;
 pf("enter a no\n");
 sf("%d",&no);
 pf("%d fact =", no);
 ~~pf("%d fact =", no);~~
 while(no>0)
 {
  f=f*no;
  no--;
 }pf("%d",f);
}
```

Scanned by CamScanner

No return value no argument

~~void fact~~

```
#include <stdio.h>
void main()
{
   void fact(void);
   fact();
}
 void fact()
 {
  int no, f=1;
  Pf("enter a no\n");
 sf("%d",&no);
 while (no>0)
 {
  f=f*no;
  no--;
 }
 Pf("fact=%d",f)
}
```

No return value with argument

```
void main()
{
 void fact(int);
 int no;
 Pf("enter a no\n");
 sf("%d",&no);
 fact(no);
}
```

Scanned by CamScanner

```
void fact (int no)
{
  int f=1;
  while (no>0)
  {
    f = f*no;
    no--
  }
  pf("fact = %d", f);
}
```

<u>return value with no argument</u>

```
void main()
{
 int fact(void);
 int res;
 res = fact();
 pf("fact = %d\n", res);
}
 int fact()
 {
  int f=1, no;
  pf("enter a no\n");
  sf("%d", &no);
 while (no>0)
 {
   f = f * no;
   no--
 }
 return f;
 }
```

Scanned by CamScanner

return value with argument

```
void main()
{
 int fact (int);
 int res, no;
 Pf ("enter a no\n");
 sf ("%d", &no);
 res = fact(no);
 pf ("fact = %d\n", res);
}
int fact (int no)
{
 int f=1;
 while (no>0)
 {
  f = f* no;
  no--;
 }
 return f;
}
```

->WAP Take a no from KB. Check no is even or odd using fun^n return value with argument.

```
void main()
{
 int check (int);
 int no;
 Pf("enter a no\n");
 sf ("%d", & no);
 if (check(no)==0)
      pf ("even no");
 else
      pf ("odd no");
}
```

Scanned by CamScanner

```
int check(int no)
{
return no%2==0?0:1;
}
```

-> Create a menthibes program. Write a program take 2 nos from KB. 1. One ~~of~~ for add^n. 2 for sub, 3 for mult using fun^n return value with argument.

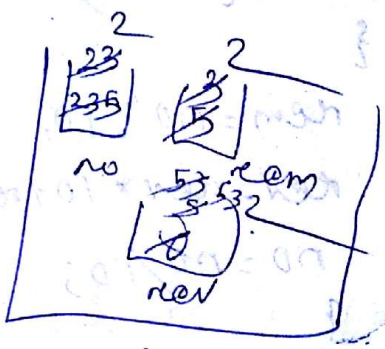
```
void main()
{
int add(int,int);
int sub(int,int);
int mult(int,int);
int no1,no2,ch;
pf("enter two nos\n");
sf("%d%d",&no1,&no2);
do
{
~~pf("\nenter your choice\n1. add(no1,no2));break;~~
~~pf("~~
pf("\nenter your choice\n1. add\n2. sub\n3. mult");
sf("%d",&ch);
switch(ch)
{
case 1: pf("sum of two no= %d\n", add(no1,no2));break;
case 2: pf("sub " " " = %d\n", sub(no1,no2));break;
case 3: pf("mult " " " = %d\n", mult(no1,no2));break;
default: pf("\n invalid choice");
}
```

Scanned by CamScanner

```
} while (ch<=3);
}
int add(int a, int b)
{
return a+b;
}
int sub (int a, int b)
{
return a-b;
}
int mul (int a, int b)
{
return a*b;
}
```

→ WAP. Take a no from KB. Find the reverse.

```
#include <stdio.h>
void main()
{
int no, rem, rev=0;
pf("enter a no\n");
sf("%d", &no);
while(no!=0)
{
rem = no % 10;
rev = rev * 10 + rem;
no = no / 10;
}
pf("rev no = %d\n", rev);
}
```



~~no~~ rem = no % 10
rev = rev × 10 + rem
no = no / 10

no return value with argument

```
#include <stdio.h>
void main()
{
  void check(int);
  int no;
  pf("enter a no\n");
  sf("%d", &no);
  check(no);
}
void check(int no)
{
  int ~~no~~ rem, rev=0;
  ~~pf("~~
  while(no!=0)
  {
   rem=no%10;
   rev=rev*10+rem;
   no=no/10;
  }
  pf("rev no = %d\n", rev);
}
```

Scanned by CamScanner

return value with argument

```
void main()
{
  int check(int);
  int no, res;
  Pf("enter a no\n");
  Sf("%d",&no);
 res=check(no);
 Pf("rev no= %d\n",res)
}
 int check(int no)
 {
  int rem,rev=0;
  while (no!=0)
  {
   rem=no%10;
   rev=rev*10+rem;
   no=no/10;
  }
  return rev;
 }
```

->WAP. Take a no from KB. Check no is palindrom

```
#include<stdio.h>
void main()
{
 int no, rem, rev=0, temp;
 Pf("enter a no\n");
 sf("%d", &no);
 temp=no;
 while (temp!=0)
```

Scanned by CamScanner

```
{
rem = temp % 10;
rev = rev * 10 + rem;
temp = temp / 10;
}
if(no = = rev)
pf ("%d is p d\n", no);
else
pf ("%d is not pd\n", no);
}
```

WAP. Take a no from KB. Check no is palendrom using fn ~~with~~ return value with argument

```
void main()
{
int no;
int check(int);
Pf("enter a no\n");
Sf("%d", &no);
if(check (no) = = no)
    pf ("%d is pd", no);
else
    Pf("%d is not pd", no);
}
int check (int no)
{
int rem, rev = 0;
while (no! = 0)
{
rem = no % 10;
rev = rev * 10 + rem;
no = no / 10;
}
return rev;
}
```

Scanned by CamScanner

PWAP Take a no from KB. Check no is perfect no. using return value with argument

```
void main()
{ int no;
  int check(int);
  pf("enter a no\n");
  sf("%d", & no);
  if (check(no) == ~~sum~~ no)
    pf("%d perfect no\n", ~~no~~);
  else
    pf("%d not perfect no\n");
}:
int check(int no)
{
  sum = 0, i;
  ~~pf(~~
  for (i = 1; i <= no/2; i++)
  { if (no % i == 0)
    {
      sum = sum + i;
    }
  }
  return sum;
}
```

Scanned by CamScanner

->WAP. take a no from KB. check no is a prime no using fun^n no return value with argument.

```
#include <stdio.h>
void main( )
{
 void check (int);
   int no;
 Pf("entere  a no\n");
 sf ("%d", & no);
 check (no);
}

 void check (int no)
 {
   int i, ctr=0;
   for (i=2; i<= no/2; i++)
    {
      if (no% i ==0)
      {
        ctr++;
        break;
      }
    }
   if (ctr==0)
 Pf ("%d preime no\n");
 else
   ("%d is not preime no\n");

}
```

Scanned by CamScanner

→ WAP. Take 2 nos from KB. Find GCD using the fun return value with argument.

```
void main()
{
int gcd (int, int);
int no1, no2;
Pf("enter two nos\n");
sf ("%d %d", &no1, &no2);
if (no1 > no2)
   Pf ("gcd = %d\n", gcd (no2, no1));
else
   Pf("gcd = %d\n", gcd (no2, no1));
}
int gcd (int n, int d)
{
 int rem = n%d;
 while (rem != 0)
 {
  n = d;
  d = rem;
  rem = n%d;
 }
 return d;
}
```

~~nos~~ res

```
if(no > no2)
{ res = gcd(no1, no2);
  Pf("gcd = %d\n", res);
  Pf("lcm = %d\n", (no1 * no2)/res)
}
else
res = gcd(no2, no1)
Pf("gcd = %d\n", res);
Pf("lcm = %d\n", (no1 * no2)/res);
```

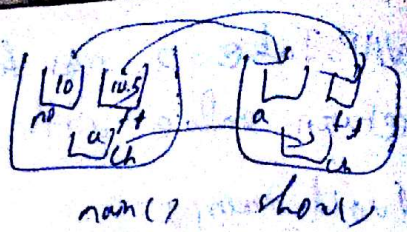
→ ~~Find GCD & LCM~~

→ WAP. initialize int, float & char variable passing into funⁿ & display it.

```
void main()
{
 int no1 = 10;
 float ~~no2~~ ft = 10.5;
 char ~~no3~~ ch = 'a';
}
```

Scanned by CamScanner

```
void show(int, char, float);
show(no, ch, ft);
}
void show(int a, char ch, float ft)
{
 pf("%d \t %c \t %-2f\n", a, ch, ft);
}
```

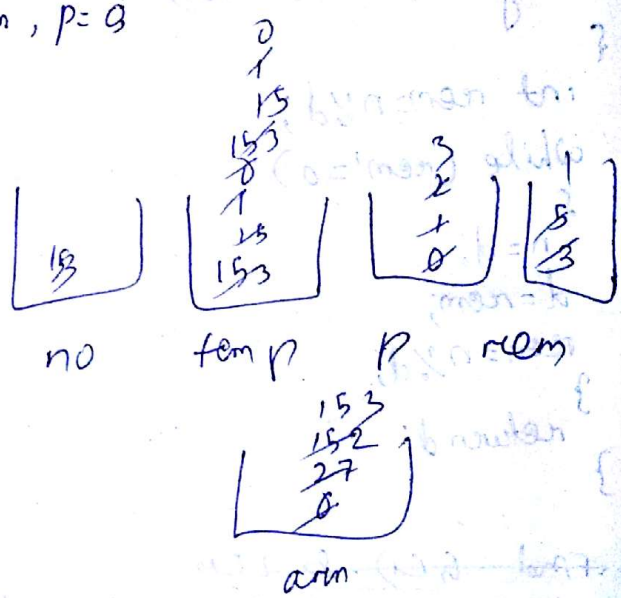


->WAP. take a no from kb check no is armstrong.

153

$3^3+5^3+1^3=153$

```
#include<stdio.h>
#include<math.h>
void main()
{
int no, temp, arm=0, rem, p=0
pf("enter a no\n");
sf("%d", &no);
temp=no;
while (temp!=0)
{
 p++;
 temp=temp/10;
}
temp=no;
while (temp!=0)
{
 rem=temp%10;
 arm=arm+pow(rem,p);
 temp=temp/10;
}
```



Scanned by CamScanner

```
if (no == arem)
pf("%d is armstrong no\n", no);
else
pf("%d is not armstrong no\n", no);
}
```

no return value with argu
return value with arg

-> Display all armstrong value upto range

No return value with argument

```
{
void check (int);
int no;
pf("enter a no");
sf("%d", &no);
check (no);
}
void chak (int no)
{
int temp, arem=0, rem, p=0;
temp=no
while (temp!=0)
{
p++;
temp=tem/10;
}
temp=no;
while(temp!=0)
{
rem=temp%10;
arem=arem+pow(rem,p);
temp=temp/10;
}
if (no==arem)
pf("%d is armstrong \n", no);
else
pf("%d is not " " " ");
}
```

Return value with arg.

```
{
int check(int);
int no;
pf("enter a no");
sf("%d", &no);
if(check (no)==no)
pf("%d is armstrong no");
else
pf("%d is not armstrong no");
```

Scanned by CamScanner

→ Display min range to max range.

```
{
 void show(int,int);
 int min,max;
 pf("enter ~~a no two~~ min & max no \n");
 sf("%d %d", &min, &max);
 show(min, max);
}
void show(int min,int max)
{
 while(min <= max)
 {
  pf("%d\t",min);
  min++;
 }
}
```

→ WAP. display all armstrong no ~~upto~~ within a range using fun^n.

```
{
void check(int,int);
int min, max;
pf("enter range min to max\n");
sf("%d %d", &min,&max);
check(min,max);
}
void check(int no, int range)
{
int temp, p=0, arm = 0, temp;
for( ; no <= range, no++)
{
temp = no;
while(temp!=0);
{
p++;
temp = temp/10;
}
temp = no;
while(temp = 0);
```

Scanned by CamScanner

Strong number

145

$= 1! + 4! + 5! = 145$

WAP. Check whether the no is strong or not.

```
{
int no, temp, rem=0, F=1, str=0;
Pf ("enter a no\n");
Sf ("%d", &no);
temp = no;
while (temp != 0)
{ rem = temp % 10;
  while (rem != 0)
  {
   f = f * rem;
   rem--;
  }
  str = str + F;
  temp = temp / 10;
  f = 1;
}
if (no == str)
Pf("strong no\n");
else
Pf("not strong no\n");
}
```

Display strong no. upto a range.

```
{
int no, temp, rem=0, F=1, str=0, range;
pf
sf
for(no=1; no<=range; no++)
{
 temp = no;
 while
 —  —
 —  —
Pf("strong no\n");
str = 0
}
```

storage class

store (no linking)

Local variable: A variable defined inside the fun^n is known as local variable. It can access inside that fun^n, can't access another fun^n.

Global variable: A variable defined outside the fun^n is known as global variable. It can access all the fun^n.

Auto storage class

auto → is a keyword. It is a default storage class of local variable. It is stored in RAM. It's default value is garbage. It's scope & life is within the block. Every time it needs reinitialization.

Syntax: storageclass datatype variable name

Scanned by CamScanner

→ void main()
{
auto int a;
pf("%d", a);
}

O/P → Garbage

→ void main()
{
int a=10;
int a=20;
pf("%d", a);
}

X redefinition of a.

→ void main()
{
int a=10;
{
int a=20;
a=30;
}
pf("%d", a);
}

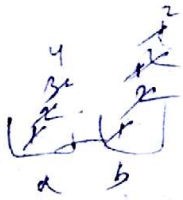
O/P → 10

→ void main()
{
{
int a=20;
}
pf("%d", a);
}

~~O/P~~ →

X 'a' undeclared

→ void main()
{
int a=1;
while (a<=3)
{
int b=1;
++a;
++b;
pf("%d %d\n", a, b);
}
}

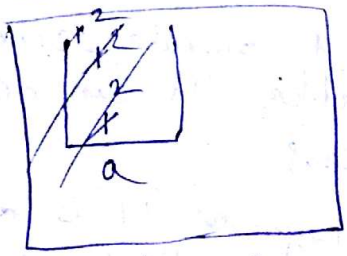


O/P → 2 2
3 2
4 2

⇒

→ If we write funⁿ deffⁿ above the main, funⁿ declaration isn't required.

→ void inc()
{
int a=1;
++a;
pf("%d\n", a);
}
void main()
{
inc();
inc();
inc();
}

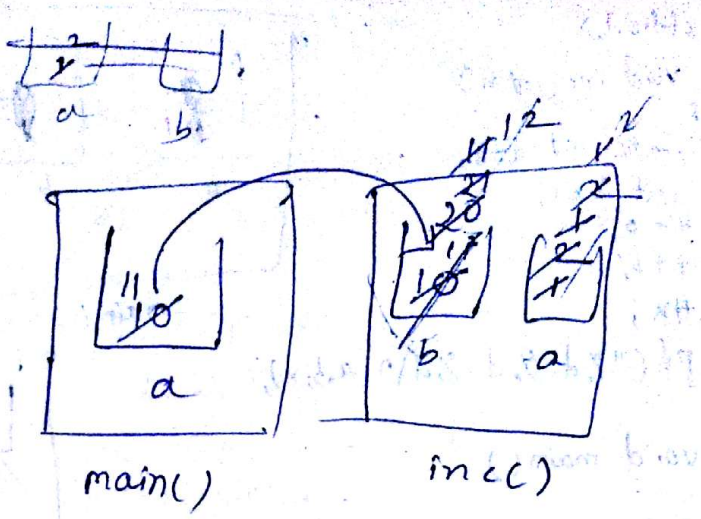


inc()

O/P → 2
2
2

Scanned by CamScanner

```
-> void inc(int b)
{
 int a=1;
 ++a;
 ++b;
 Pf("%d %d\n",a,b);
}
void main()
{
 int a=10;
 inc(a);
 ++a;
 inc(20);
 inc(a);
 Pf("%d\n",a);
}
```



O/P -> 2 11
2 21
2 12

11

<u>static storage class</u> -> is created in RAM. default value 0.
We can use it in two ways -> 1. Local static (Avoiding reinitialisation)
2. Global static (file scoping)

```
-> void main()
 {
  static int a;
  Pf("%d",a);
 }
```

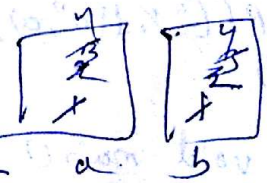
O/P -> 0

```
-> void main()
 {
  {
  static int a;
  }
  Pf("%d",a);
 }
```

X a is undeclared
a life is available but we can't access that location

```
-> void main()
 {
 int a=1;
 while(a<=3)
 {
  static int b=1;
  ++a; ++b;
  Pf("%d %d\n",a,b);
 }
}
```

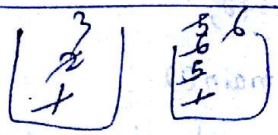
-> line is avoided from 2nd turn execution



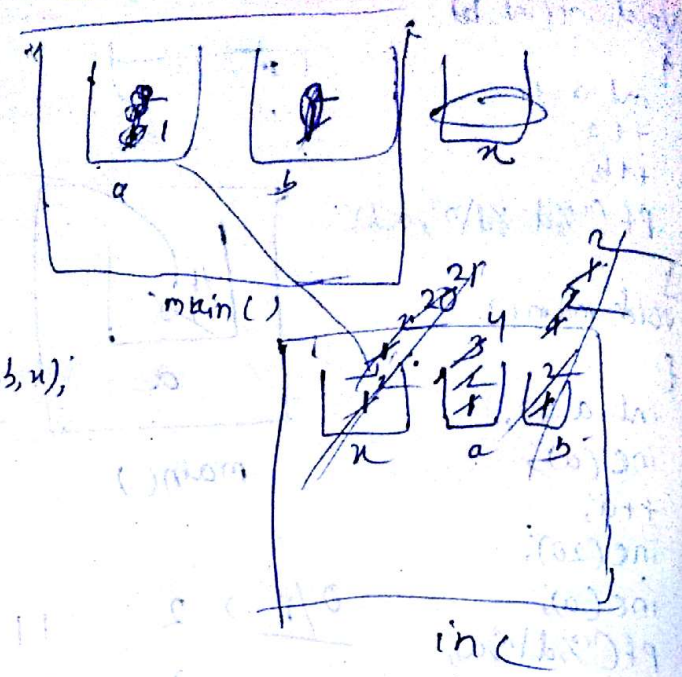
O/P -> 2 2
3 3
4 4

```
-> void main()
 {int a=1;
 while(a<=3)
 {
 static int b=1; b=5;
 ++a; ++b;
 Pf("%d %d\n",a,b);
 }
}
```

O/P -> 2 6
3 6
4 6



```
#include<stdio.h>
int b=1;  void inc(int x)
++a;      {
++b;       static int a=1;
++x;       int b=1;
           ++a;
           ++b;
           ++x;
           pf("%d %d %d\n",a,b,x);
          }
          void main()
          {
           int a=1;
           inc(a);
           inc(a);
           inc(20);
           pf("%d\n",a);
          }
```



O/P →

2	2	2
3	2	2
4	2	21

int a; → Variable deff

extern int a; → Variable declaration

→ void main()
{
extern int a; ✗ linking error
pf("%d\n", a); Undefined reference to a.
}

→ void main()
{
extern int a=10; ✗ compilation error
pf("%d\n", a); extern variable can't be declared
} & with initialisation.

→ void main()
{
pf("%d\n", a); ✗ a is undeclared
}
int a=10;

→ void main()
{
extern int a; → declaration
pf("%d\n", a);
}
int a=10; → definition

internal linking - variable declaration & definition in same file.

Scanned by CamScanner

```
void main()
{
extern int a;
    void show(void);
    pf("%d\n",a);
    show();
    pf("%d\n",a);
}
 void show()
 {
  pf("%d\n",a);
  a = 40;
 }
 int a=10;
```

```
void main()
{
extern int a;
    void show(void);
    pf("%d\n",a)
    show();
    pf("%d\n",a);
}
void show()
{
extern int a;
pf("%d\n",a);
a = 40;
}
int a=10;
```

O/P: 10
10
40

→ If we variable deff^n above the main, variable declaration isn't required.

→ Inside the fun^n, fun^n declaration's storage class is extern by default.

```
void main()
{
 void bbsr(void);
 pf("mfs\n");
 bbsr();
 pf("mfe\n");
}
void bbsr()
{
pf("bfs\n");
pf("bfe\n");
}
```

→ If we write fun^n declaration above the main, any fun^n call ~~a~~ any fun^n directly.

Scanned by CamScanner