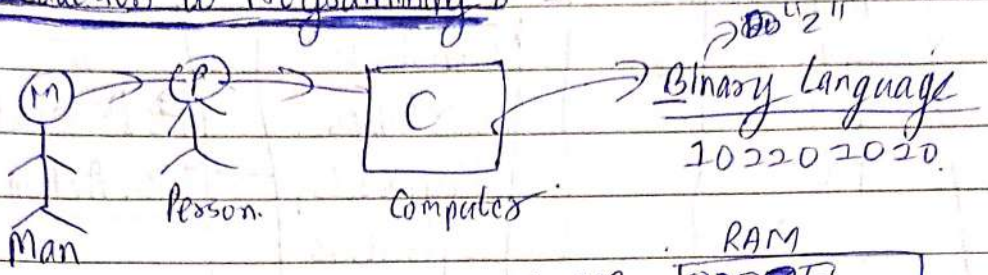
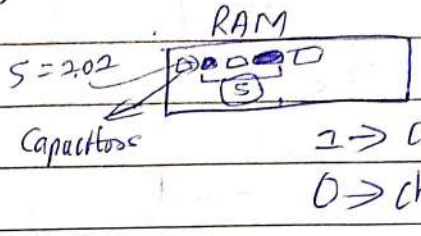
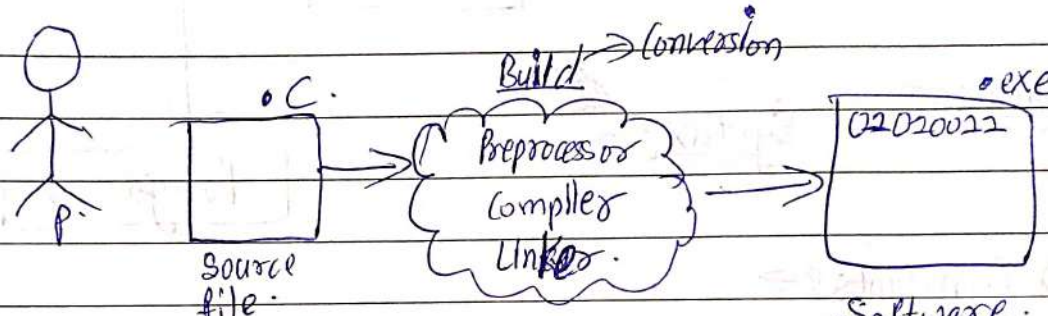
# C Programming Language

## *) Introduction to Programming :→




Steps

① Create a Source file.

② Build software.

## *) History :→

→ BCPL :→ Basic combined Programming language.
↳ 1966 → Martin Richards.

→ B → 1969 → Ken Thompson.

→ C → 1972 → Dennis Ritchie → High level as well as Low level language.

→ Unix

→ AT & T's Bell Labs, USA.

Date : / /
Page :

# Begin "C"

Basics of languages
- A to Z
- a to z
- 0 to 9
- $, [, ;
- ?, !, ~
- :, ;
- >, <, +
- /, \, ()

Analogy with human languages

Tokens → just like english words
- Constants
- Variables
- Keywords
- Operators
- Punctions
- =

Instruction → statements in language
1. Decloration statements
2. Action statements

Grammar ↔ Syntax :-> Program

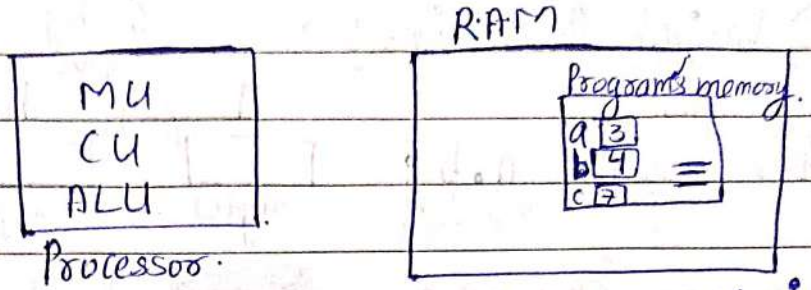
## *) Constants :->

Data = Information = Constant

### Primary Constants :->
1. Integer -> -5, 25, 42, etc
2. Real -> 3.7, 3.0, -0.02, etc.
3. Character -> 'a', 'A', '+', etc
   - Single quotes: '-3' -> X, '35' -> X, '4.2' -> X
   - Length is more than "1".

### Secondary Constants :->
- arrays
- Strings -> "BHOPAL"
- Pointers
- Structure
- Unions
- enumeration

## *) Variables :->




a, b, c are Variables.

## # Variable naming Rules :->

-> Variable name is any combination of alphabet, digit and underscore.
-> No other symbol is allowed.

| | |
|---|---|
| a -> ✓ | a$ -> X |
| a25 -> ✓ | A.b -> X |
| a-2-b-3 -> ✓ | x-1 -> X. |

-> Valid variable name cannot start with a digit.

## *) Keywords :-> Online dekhlo ✓

## *) Data types :->

1. Size required to store data.
2. internal binary representation of data.
3. Kind of operations.

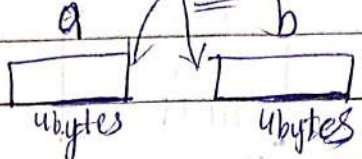
| Primitive | Non-Primitive |
|---|---|
| int | ↓ |
| char. | Khud se banaya hua data type. |
| float. | |
| double. | |
| void | |

## #) Variable Declarations :=>

Garbage → Jo memory jab kch value nhi rkha ho tab wo garbage value rakh leta hai.

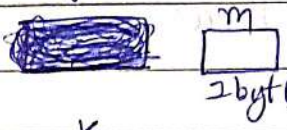
Memory consumption

4 bytes ⟵ int a, b;  (4 byte, 4 byte.)




Integer constant.

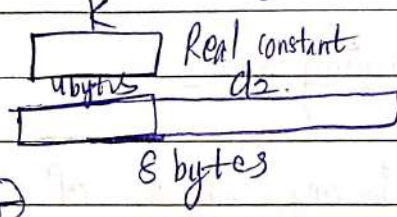
1 bytes ⟵ Char m;




Character constant

4 bytes ⟵ float K;

8 bytes. ⟵ double d2;



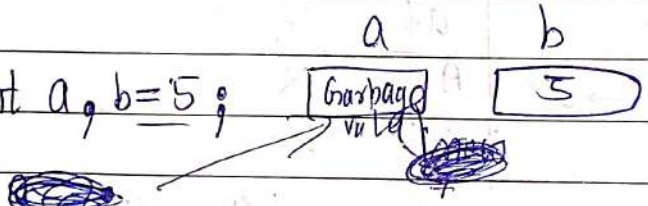

Real constant

Real constant

⊕

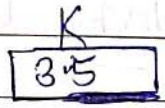
4+4+1+4+8 = 21 bytes

int a, b=5;




Char m = 'A';


float K = 3.5;

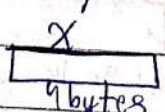



## #) float v/s double :=>

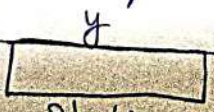
float → 4 bytes.

double → 8 bytes

float x = 0.7;




double y = 0.7;




Binary conversion :=>

0.7 ⇒ 0.7 × 2 = 1.4 = 1

0.4 × 2 = 0.8 0

0.8 × 2 = 1.6 1

0.6 × 2 = 1.2 1

0.2 × 2 = 0.4 0

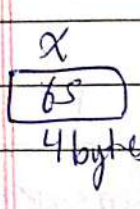
0.4 × 2 = 0.8 0

0.8 × 2 = 1.6 1

0.7 ⇒ 1011001100...

#) int v/s char :->

```
char m = 65;
char m = 'A';
int x = 'A';
int x = 65;
```




Character encoding :-> ASCII.

256 characters
codes = 0 to 255.

| | | | |
|---|---|---|---|
| Space | ' ' 32 | @ | 64 |
| | '0' 48 | 'A' | 65 |
| | '1' 49 | 'B' | 66 |
| | ⋮ | ⋮ | |
| | '9' 57 | 'Z' | 90 |
| | | 'a' | 97 |
| | | 'b' | 98 |
| | | ⋮ | |
| | | 'z' | 122 |

255
↳ Binary
1 1 1 1 1 1 1 1
1 byte = 8 bit

*) Block Structure :-> { } -> Block.

-> C is a block structured programming language.
-> A block is a group of instruction.
-> Outer blocks are usually functions.

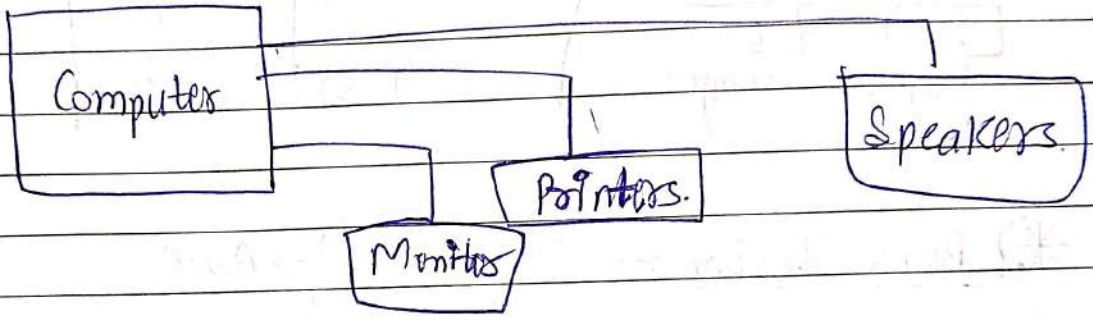
-> Function is a block of statements, which has some name for identification.
-> A 'C' program can have any number of blocks.
-> Even in the smallest 'C' program, there is atleast one function.
-> If there is only one function in the program then its name must be main().
-> function names must be unique.
-> You can write declaration statements outside the function body, but action statements must be written inside the function body.

Date : / /
Page :

```
int a, b ; ⟶ Global Variable
main ()
{
   int x ; ⟶ local Variable
}
```

## ✱) Output Instructions :⇒

Output Devices




⟶ Printf() → Predefined Function.

- printf is used to print a text on the monitor.
- printf can be used to print value of a variable or an expression.

```
printf ("Welcome") ;
```

Action Statements :⇒

(1). I/O instruction
(2). Arithematic Instruction
(3). Control Instruction.

Q). Write a program to print Hello Students on the screen.

⇒

```
#include<stdio.h>
#include<conio.h>
int main ()
{
   printf ("Hello Students") ;
   getch () ;
}
```

getch() (conio.h)
↓ ↓
Lena Character
keyboard se.

⟶ Screen pakne ke use mein aata hai

# Escape Sequences :⇒

⇒ '\n' → New Line
'\t' → Tab Space.
'\b' → BackSpace.
'\r' → Carriage Return.
'\\' → Print \
'\"' → Print "
'\'' → Print '

Q.) How to print value of a variable?

⇒ int a=5;
printf("a"); → a ✗
printf("5"); → 5 ✗
printf("%d", a); → 5 ✓

printf("a=%d", a);
a = 5

printf("Value of a is %d", a);
Value of a is 5.

printf("%d, %d", a); ✗

printf("%d %d", a, a); ✓
5 5

printf("%d %d %d", a, a*a, a*a*a);
Expression
5 25 125.

## Format Specifier. :⇒

%d → int.
%c → char.
%f → float.
%lf → double.

## *). Taking Input through Keyboard :=>

### #). Input Instruction :=>

*). **getch()** :=> It can take only one character at a time.
↳ Syntax :=> `x = getch();`

*). **scanf()** :=> → It can take data which require multiple Keystrokes.

→ It can take data as a sequence of characters and uses space, tab and new line character as data separater.

→ It can convert data into desired type.

→ It can store data in specified variable.

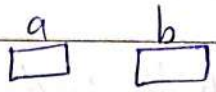
→ **Example** :=>

```
main()
{
    int a;
    scanf("%d", &a);
    printf("a=%d", a);
    getch();
}
```


& → 'address of'
= ↳ 'Referencing' operator

→ Example to take multiple values :=>

```
main()
{
   int a,b;
   scanf("%d %d", &a, &b);
   printf("a= %d  b= %d", a, b);
   getch();
}
```




Q>. Write a program to calculate sum of two numbers.

Soln =>

```
#include<stdio.h>
#include<conio.h>

int main()
{
   int a, b, c;
   printf("Enter two numbers");
   scanf("%d %d", &a, &b);
   c = a+b;
   printf("Sum of %d and %d is %d", a, b, c);
   getch();
}
```

Date : / /
Page :

## *) Operators in "C" language :⇒

## #) Arithematic Instruction :⇒

→ An Instruction which is used to manipulate data using operators, is Known as arithemetic Instruction.

operands/data.

3 + 4

operator.

3+4*5 = 23 ✓

BODMAS.

② Unary operators
② Binary operators
③ Ternary operators

## #) Operators :⇒

① Unary operators
② Arithemetic operators
③ Bitwise Operators.
④ Relational Operators.
⑤ Logical Operators.
⑥ Conditional Operators
⑦ Assignment Operators.

Priority Order

There is no BODMAS Rule in "C".

## ②) Unary Operators :⇒

+, -, ++, --, sizeof().

→ int x=5;

printf("%d", x); →5

Post increment → x++; → x = x+1. → Priority ↓.

printf("%d", x); → 6

Pre-Increment → ++x; → x = x+1. → Priority ↑

printf("%d", x); → 7

Q. main()
{
int x=5, y;  x → 6, y → 5
Post Increment y=(x++);
Priority.
printf("%d %d", x, y); → 5 6 ✗
}
6 5 ✓

⇒ x--; Post decrement
--x; Pre decrement
x = x-1

⇒ double y;
x = sizeof(y);
↓
8 bytes.

⇒ sizeof( )
(1) Datatype.
(2) Variable.
(3) Constant.

Type modifiers ⇒
short
long
Eg ⇒ short int x;
short x;

## Arithmetic Operators ⇒ (Left to Right).

* / % → B तीनों का Priority Same hai "C" mein.
+ - → But inn dono se jyada hai.

⇒ a*b/c. → * → Ko solve Kro.
a/b*c → / → "
a+b*c → * → Priority "+" se jyada hai.

3+4=7
3-4=-1
3*4=12
3/4 = 0 → Rule hai "C"-mein ki Jab aap do Integer se khel rhe ho tab answer Integer hee hoga.

3.0/4 → 0.75 → Pr vi real hoga toh answer Real aayega.
3/4.0 → 0.75
3.0/4.0 → 0.75.

→ 35 (%) 6 → modulus remainder batata hai.

ans = (5)

2.5 % 3 → error → "C" mein real constant pr modulus operator apply nhi kar skte.

2 % 5 → 2

5) 2 (0
   0
   2

#) Bitwise Operator ⇒

| & AND | \| OR | ^ XOR | ~ NOT | >> Right Shift | << Left Shift. |
|---|---|---|---|---|---|
| 0&0 → 0 | 0\|0 → 0 | 0^0 → 0 | ~0 → 1 | | |
| 0&1 → 0 | 0\|1 → 1 | 0^1 → 1 | ~1 → 0 | | |
| 1&0 → 0 | 1\|0 → 1 | 1^0 → 1 | | | |
| 1&1 → 1 | 1\|1 → 1 | 1^1 → 0 | | | |

int x;
x = 47 & 29;

# Trick for Binary conversion ⇒

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

29 ⇒ 0 0 0 1 1 1 0 1 | 47 ⇒ 0 0 1 0 1 1 1 1

Date / Page

| | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|
| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 47 = | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 29 = | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| & operation ⇒ | | | | | | | | |
| 13 = | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

## ⇒ Right & left Shift :⇒

```
x = 206 >> 2;
```

Do bago shift.

206 → int type ⇒ 4 bytes ⇒ 32 bits.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 0

24 bits → 0 0

Remaining 8 bits

| | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|
| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |

= 26

## #) Relational Operators :⇒

< > <= >=

== !=

| |
|---|
| True = 1 |
| False = 0. |

3 < 4 ⇒ 1 ⇒ True.

5 >= 5 ⇒ 1 ⇒ True.

2 == 3 ⇒ 0 ⇒ False.

(5 > 4) > 3 ⇒ 0

L to R

1 > 3

## #) Logical Operators :=>

! => NOT (unary)
&& => AND.
|| => OR.

True = 1
False = 0

!T => F.
!F => T.

=> 5 > 4 => 1.
!(5>4) => 0

x > 2 && x < 5

T && T => T.
T && F => F.
F && X => F
↳ Koi vi cheej
Ek vi false mtlb result false.

F || F => F.
F || T => T.
T || X => T
↳ Anything

## #) Assignment Operators :=>

=, +=, -=, *=, /=, %=.
Compound assignment operators

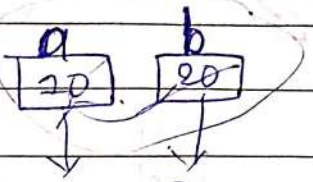
x = 4;
4 = x; ✗
Variable = ? ✓

=> int x = 5;
x += 2; => x = x + 2.
x *= 3; => x = x * 3
x %= 4; => x = x % 4

## Decision Control Instruction

Q). Write a program to swap values of two int type variables.

Sol" => printf("%d %d", b, a);



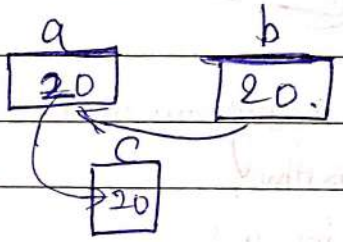

20 20

X

Wrong Logic

```
a = b;
b = a;
```

X




Right Logic

```
c = a;
a = b;
b = c;
```

### * Control Instruction :=>

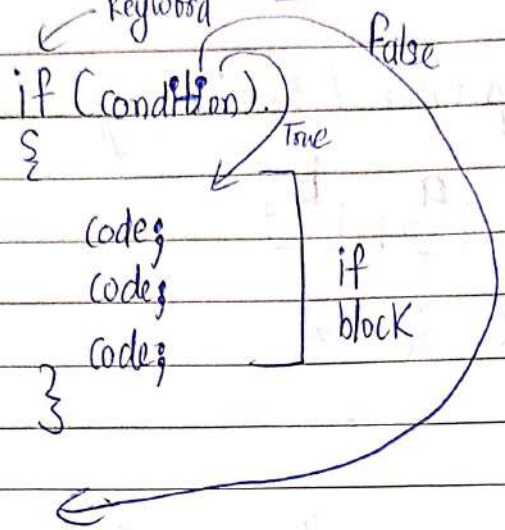
1. Decision Control Instruction.
2. Iterative control Instruction.
3. Switch case control Instruction.
4. goto Control Instruction.

### #(1) Decision Control Instruction :=>
(Selection Control Instruction).

1. if
2. if else.
3. ?: (Conditional operator).

Date : / /
Page :

② if

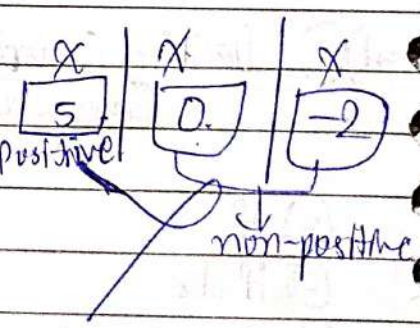



*) Rule :-
#) Every Non-zero value is True & Zero is False.

Q.) Write a program to check whether a given number is positive or non-positive.

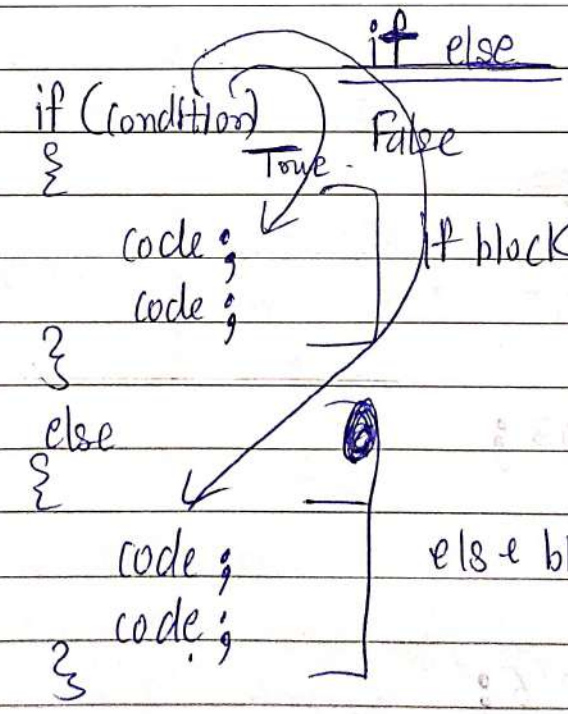
Sol^n =>

```
int a;
if ( a>0 )
{
printf ("This number is positive", a);
}
printf ("This is not positive", a);
main()
{
int x;
printf ("Enter a number");
scanf ("%d", &x);
if (x>0)
{
printf ("Number is positive");
}
printf ("Number is nonpositive");
getch();
}
```



```
main()
{
    int x;
    printf("Enter a number");
    scanf("%d", &x);
    if (x>0)
    {
        printf("Positive");
    }
    if (x<=0)
    {
        printf("Non-Positive");
    }
    getch();
}
```

*)



Q.) Previous program :->

```
main()
{
    int x;
    printf("Enter a number");
    scanf("%d", &x);
    if (x>0);
    {
        printf("Positive");
    }
    else
    {
        printf("Non-Positive");
    }
    getch();
}
```

#) Conditional Operators :->
(Ternary Operator)

(? :)

Requires "3" action statements

exp1 ? exp2 : exp3 ;

Always a condition.

True

False.

y = x>0 ? 5 : 6;

Q). Previous Program ⇒

```
main()
{
    int x;
    printf("Enter a number");
    scanf("%d", &x);
    x>0 ? printf("Positive") : printf("Non-Positive");
    getch();
}
```

(M-2) ⇒

```
printf(x>0 ? "positive" : "non-positive");
```

Q). Write a program to check whether a given number is divisible by 5 or not.

⇒

```
int main()
{
    int x;
    printf("Enter a number");
    scanf("%d", &x);
    if (x%5==0)
    {
        printf("Divisible by 5");
    }
    else
    {
        printf("Not divisible by 5");
    }
    getch();
}
```

Better Soln ⇒

```
if (x%5)
{
    printf("Not Divisible by5");
}
else
{
    printf("Divisible by 5");
}
```

x = 25
divisible by 5

x = 23
not divisible by 5

x%5 → 0 → Treated as False
x%5 → 1, 2, 3, 4 → Treated as True.

Q.) Write a program to check whether a given number is even or odd.

Sol^n =>

```
main()
{
  int x;
  printf("Enter a number");
  scanf("%d",&x);
  if(x%2)
  {
    printf("Odd no.");
  }
  else
  {
    printf("Even number");
  }
  getch();
}
```

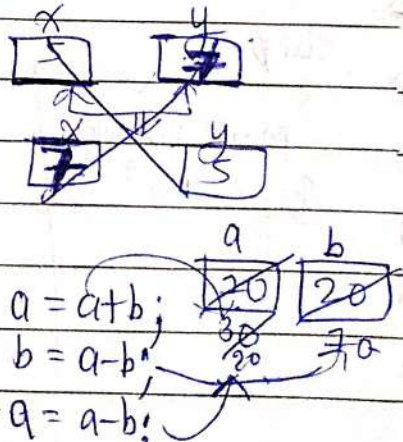
x%2 → 0 (if divisible by 2) → false → This jumps to else block

x%2 => 1,2,3,4 → True → Remains in "if" block.

Q.) Write a program to swap values of two int variables without using third variable.

Sol^n =>

```
main()
{
  int x,y;
  printf("Enter values of ");
  scanf("%d %d",&a,&b);
  a = a+b;
  b = a-b;
  a = a-b;
  printf("Value of ",a);
  getch() printf("value of ",b);
}
```



Q.). How to find last digit of a given number?

Soln:

```
main()
{
int x;
printf("~~");
scanf("%d", &x);
if (x%10)
{
printf
}
else
{
printf
}
getch();
}
```

Logic =>

x = 253

3

x % 10 ↑

```
main()
{
int x;
printf("Enter a no.");
scanf("%d", &x);
x = x%10;
print("The last digit is", x);
getch();
}
```

Q). How to remove last digit from a given number?

Soln =>

x % 10 → 3 ✗ ← x = 253

x/10 → 25 ✓

Output = 25

x = x/10; ✓

or

x /= 10; ✓

## *) Nested if / Nested if-else :=>

(1) if (condition)
{
    if (condition)
    {
        ≡
    }
}

(2) if (condition)
{
    if (condition)
    {
        ≡
    }
    else
    {
        ≡
    }
}

(3) if (condition)
{
    if (condition)
    else
}
else
{
    if ((condition)
    else
}

## *) "if-else" Ladder :=>

if (condition). ——→ (1)
else if (condition). ——→ (2)
else if (condition) ——→ (3)
else ——→ (4).

inn 4 mein se exactly sirf ek hee chalega.

## *) Iterative Control Instruction (LOOP) :=>

Q.) Write a program to check whether a given number is divisible by 3 and 2.

Sol^n =>
```
main()
{
    int x;
    printf("Enter a number");
    scanf("%d", &x);
    if (x%3==0 && x%2==0)
        printf("Not Divisible by 3 & 2");
    else
        printf("Not Divisible by 3 & 2");
    getch();
}
```

Q.) Write a program to check whether a given number is divisible by 7 or divisible by 3.

Sol^n =>
```
main()
{
    int x;
    printf("Enter the value");
    scanf("%d", &x);
    if (x%7==0 || x%3==0)
        printf("Divisible by 7 or 3");
    else
        printf("Not Divisible");
    getch();
}
```

Q.) Write a program to check whether a given character is an alphabet (upper case), an alphabet (lower case), a digit or a special character.

Sol"=>

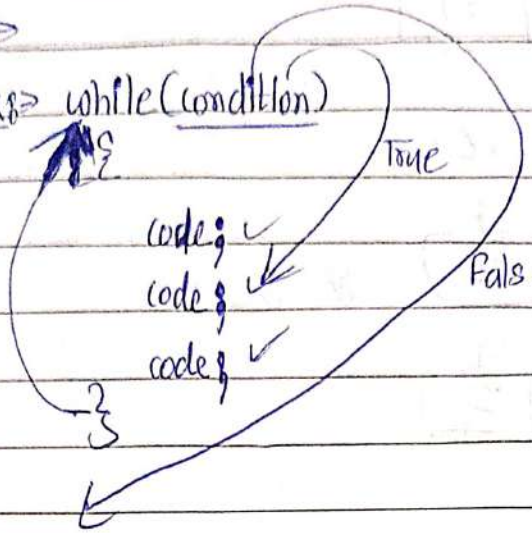
```
main()
{
    char x;
    printf("Enter a character");
    scanf("%c", &x);
    if (x>='A' && x<='Z')
        printf("Upper case Alphabet");
    else if (x>='a' && x<='z')
        printf("Lower case Alphabet");
    else if (x>='0' && x<='9')
        printf("Digit");
    else
        printf("Special Character");
    getch();
}
```

*) Iterative Control Instruction :=>

1.) while
2.) do while
3.) for.

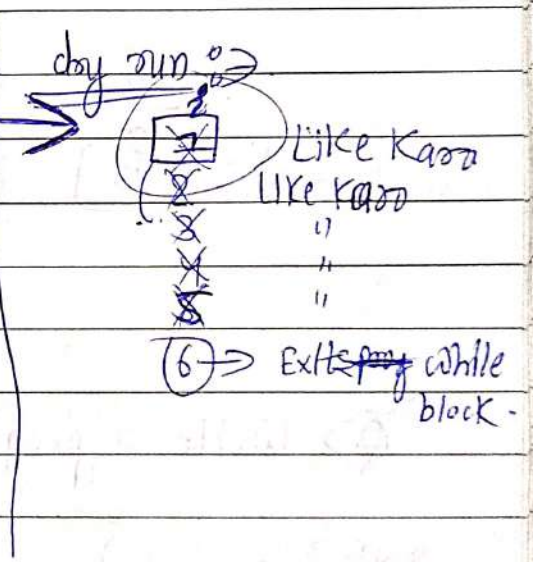
1.) While :=>

Syntax:=> while (condition)
{
code;
code;
code;
}



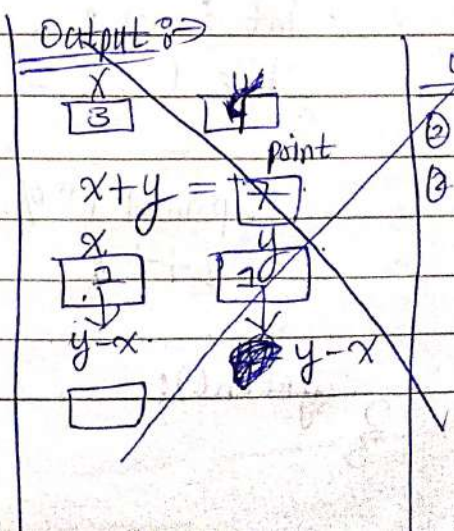

Q.) Write a program to print "LIKE KARO" five times on the screen.

Soln=>
```
main()
{
  int i=1;
  while (i<=5)
  {
    printf ("LIKE KARO");
    i++;
  }
  getch();
}
```

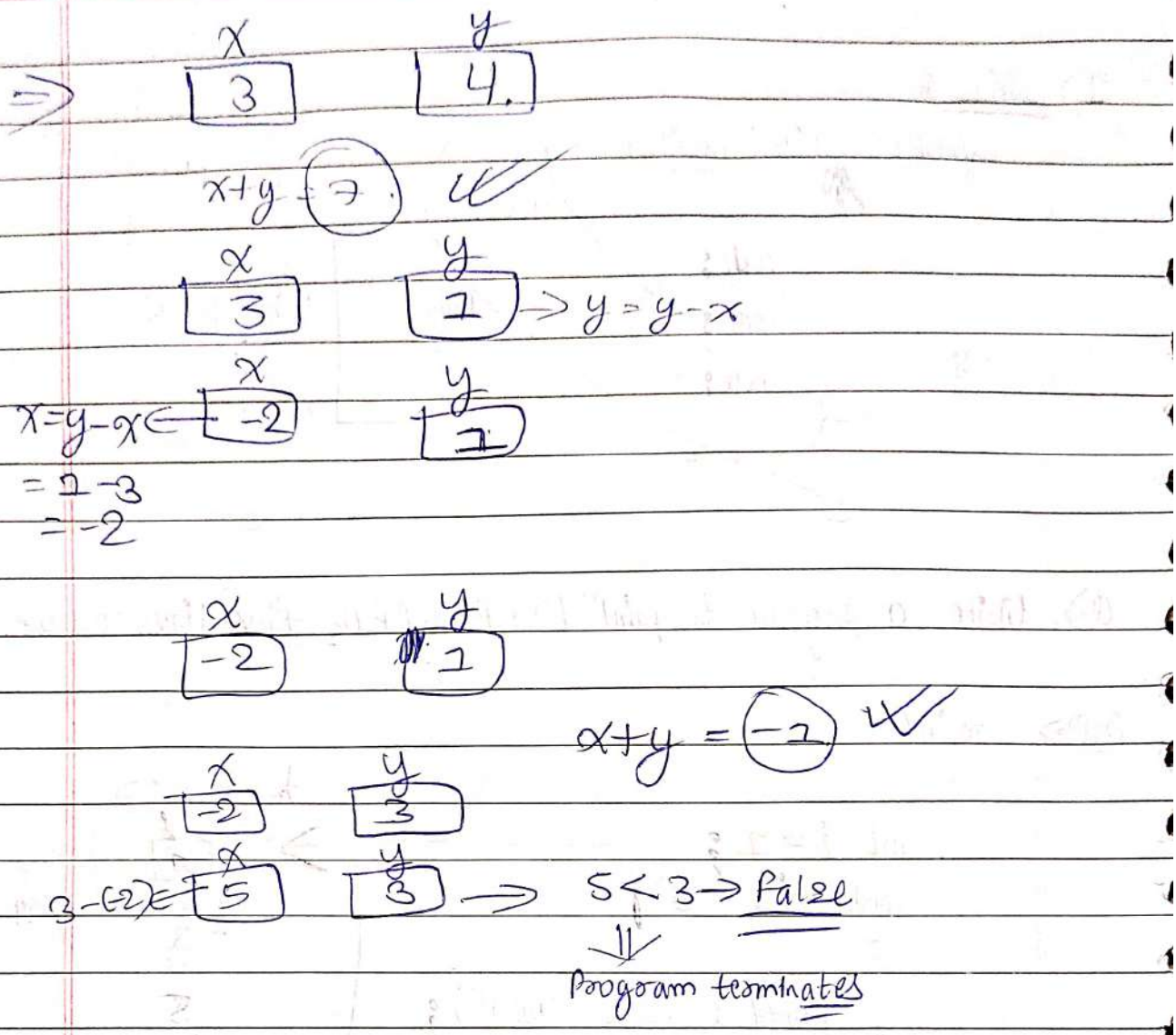



Q.) Find Output :=>

Soln=>
```
main()
{
  int x=3, y=4;
  while (x<y)
  {
    printf ("%d", x+y);
    y = y-x;
    x = y-x;
  }
  getch();
}
```



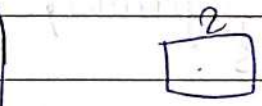
Date : / /
Page :




Q.) Write a program to print first 10 natural numbers.

Sol:

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




1 <= 10 → T
2 <= 10 → T
3 <= 10 → T
⋮
9 <= 10 → T
10 <= 10 → T
11 <= 10 → F

Q) Find Output :=>

```
main()
{
    int i=20;
    while(i)
    {
        printf("%d", i);
        i-2;
    }
    getch();
}
```

| i | Output |
|---|---|
| 20 | 20 |
| i-2 = 20-2 = 9 | 9 |
| | 8 |
| | 7 |
| | 6 |
| B'coz => i = i-2 is not written | 4 |
| | 3 |
| | 2 |
| | 1 |
| | 0 X |

Every Non-zero → True.
zero → False.

↳ 20 20 2020 20 20 ---- ∞ → Infinite Times

Q) Write a program to print first 20 even natural numbers.

Output :=> 2, 4, 6, 8, 20, 22, 24, 26, 28, 20

=>

```
main()
{
    int i=1;          // 2
    while(i<=20)      // 20
    {
        printf("%d", 2*i);
        i=i+1;        // i=i+2;
    }
    getch();
}
```

mera method bhi sahi tha.

| i | 2*i |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 20 |
| 6 | 22 |
| 7 | 24 |
| 8 | 26 |
| 9 | 28 |
| 20 | 20 |

Q. Find Output :→

```
main()
{
    int i=1;
    while (i<=20)
    {
        printf("%d", 11-i);
        i++;
    }
}
getch();
```

Output :→

i
1

11-i

11-1=10
11-2=9
11-3=8
11-4=7
11-5=6
11-6=5
11-7=4
11-8=3
11-9=2
11-10=1

Output

11<=10 → False

Q. Write a program to print first N odd natural numbers in reverse order.

Soln →

```
main()
{
    int n=1;
    while(
    {
    printf("%d",
```

a=-2
9 7 5 3 1 - - -
Tn = a+(n-1)d
= 9+(n-1)(-2)
= 11-2n

eg :→
N = 5
1 3 5 7 9
9 7 5 3 1

11-2n

```
main()
{
    int n, i=1;
    printf("Enter a number");
    scanf("%d", &n);
    while (i<=n);
    {
        printf("%d", 2*n+1-2*i);
        i++;
    }
}
```

| 2*i |
|---|
| 2 |
| 4 |
| 6 |
| 8 |
| 10 |

| i | reverse output |
|---|---|
| 1 | 9 |
| 2 | 7 |
| 3 | 5 |
| 4 | 3 |
| 5 | 1 |

| 2n+1 - 2*i |
|---|
| 9 |
| 7 |
| 5 |
| 3 |
| 1 |

Date : / /
Page :

## While

Syntax :=>

```
while (condition)
{
   ==
   ==
}
```

e.g :=> int i=1;

```
while (i<=10)
{
   print ("%d", i);
   i++
}
```

dry run :=>

1<=20 → T
2<=20 → T
3<=20 → T
⋮
20<=20 → T
22<=20 → F
↳ Exits

10 Times → True
1 Time → False

*) While loop :=>
Entry control loop
↓
while mein koi garauntee nhi ki code ek baar vi chalega ya nhi.

## do-while

Syntax :=>

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

(True / False)

e.g :=> int i=1;

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

dry run :=>

1<=20 → T
2<=20 → T
3<=20 → T
4<=20 → T
⋮
9<=20 → T
20<=20 → T
22<=20 → False

9 Times → True.
1 Times → False.

*) Do-While loop :=>
Exit control loop.
↓
code atleast 1 Time chalega hee.

## for

Syntax :=>

```
for ( ; ; )
{
   ==
   ==
}
```

e.g :=> int i;

```
for (i=1; i<=20; i++)
{
   printf("%d", i);
}
```

(True / False)

dry run :=>

1<=20 → T
2<=20 → T
3<=20 → T
⋮
9<=20 → T
20<=20 → T
22<=20 → F

20 Times → True
1 Time → False

*) for loop :=>
Entry control loop
↓
pehle condition check hoti hai phir block ke andar aati hai.

Q) Write a program to calculate sum of first "n" natural numbers.

Sol :=>

```
main()
{
   int n, i=1;
   printf("Enter a value");
   scanf("%d", &n);
   while (i<=n)
   {
      printf("%d", i)
      i++;
   }
   printf("Sum is", n*(n+1)/2);
   getch();
}
```

dry run :=>

n = 5 (first five natural nos.)

| i | Sum = i(i+1)/2 |
|---|---|
| 1 | first "n" natural number = n(n+1)/2 |

Date : / /
Page :

Sol ⇒

```
main()
{
    int n, i, s=0;
    printf("Enter a number");
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
        s = s+i
        printf("Sum is %d", s);
    }
    getch();
}
```

Dry run :→

n = 5

1+2+3+4+5 = 25

s = 0

① s = s+1 (i = 1)
② s = s+2 (i = 2) } s = 3
③ s = s+3 (i = 3)
④ s = s+4 (i = 4) } s = 10
⑤ s = s+5 (i = 5) ⇒ s = 15

## *) break :→

→ "break" is a Keyword.
→ "break" can be used either in loop's body or switch's body.
→ In loop, break is used to terminate execution of loop.

Syntax ⇒

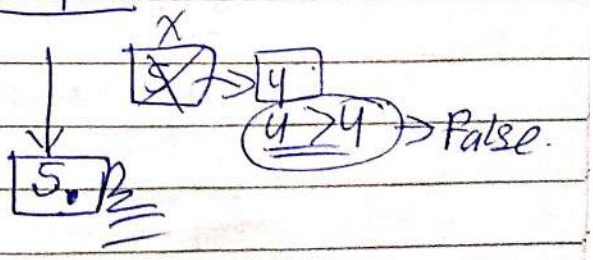
```
int i=1;
while(age<=100)
{
    ==
    if(--)
        break;
    ==
    age++;
}
```

Q). Define a game like program, in which user has to enter an even number to win the game. User will get at most 3 chances.

Soln=>
```
main()
{
    int x, i;
    for (i=1; i<=3; i++)
    {
        printf("Enter an even number");
        scanf("%d", &x);
        if (x%2==0)
            break;
    }
    if (i==4)
        printf("Game over");
    else
        printf("you win");
    getch();
}
```

Q). Find Output :=>
```
main()
{
    int x=5;
    do
    {
        printf("%d", x);
        x--;
    } while (x>4);
}
```

Output :=>



Q). Find Output :⇒

```
main()
{
    int i, j;
    for (i=1, j=10; i<j; i++, j--)
    {
        if (i==j)
            break;
        printf("%d", i+j);
    }
    getch();
}
```

Output :⇒

i  j

1 < 10 ↳T = 11

2 < 9 ↳T = 11

3 < 8 ↳T = 11

4 < 7 ↳T = 11

5 < 6 ↳T = 11

6 < 5 ↳ False

5 times → 11 will be printed

*). Nested Loops :⇒

1). while ( ) ← 5

{

for ( ; ; )

{

⋮ ← 10

}

⋮

}

50 times

2). for ( ; ; )

{

for ( ; ; )

{

⋮

}

⋮

}

Q.). Write a program to generate table chart :=>

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | | | 20 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | | | 20 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | | | 30 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | | | 40 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | | | 50 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | | | 60 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | | | 70 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | | | 80 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | | | 200. |

Soln=>

```
for (i=1; i<=20; i++)
{
    for (j=1; j<=20; j++)
    {
        printf ("%d ", j*i);
    }
    printf ("\n");
}
```

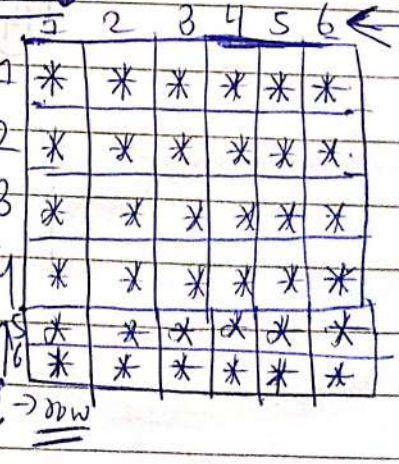
Output

| i | Output |
|---|---|
| 1 | 1 2 3 4 5 6 7 8 9 20 |
| 2 | |
| 3 | |
| . | |
| . | |
| . | |

*) Star Pattern (Basic Skeleton) :=>

~~Basic~~ General Trick :=> (works at max^m cases)

```
for (i=1; i<=6; i++)
{
    for (j=1; j<=6; j++)
    {
        printf ("*");
    }
    printf ("\n");
}
```



Date : / /
Page :

② ⇒
```
for (i=1; i<=6; i++)
{
    for (j=1; j<=6; j++)
    {
        if (j<=7-i)
            printf("*");
        else
            printf(" ");
    }
    printf("\n");
}
```

| | 1 | 2 | 3 | 4 | 5 | 6 ← j |
|---|---|---|---|---|---|---|
| 1 | * | * | * | * | * | * |
| 2 | * | * | * | * | * | |
| 3 | * | * | * | * | | |
| 4 | * | * | * | | | |
| 5 | * | * | | | | |
| 6 | * | | | | | |

↑ i

"*" ⇒ 22 times print

"space" ⇒ 28 times print

Steps ②

| i | j |
|---|---|
| 1 | 1 2 3 4 5 6 |
| 2 | 1 2 3 4 5 |
| 3 | 1 2 3 4 |
| 4 | 1 2 3 |
| 5 | 1 2 |
| 6 | 1 |

③

| i | j |
|---|---|
| 1 | j<=6 |
| 2 | j<=5 |
| 3 | j<=4 |
| 4 | j<=3 |
| 5 | j<=2 |
| 6 | j<=1 |

④ j <= 7 - i

Q). #Include <stdio.h>
#include <conio.h>
int main()
{
int i, j, lines;
printf("Enter number of lines");
scanf("%d", &lines);

for (i=1; i<=lines; i++)
{
for (j=1; j<=lines; j++)
{
if (j<=lines+1-i)
printf("*");
else
printf("");
}
printf("\n");
}
getch();
}

Q). main()
{
int i, j, lines, K;
printf("Enter number of lines: ");
scanf("%d", &lines);

for (i=1; i<=lines; i++)
{
→ for alphabets K='A';
K=1;
for (j=1; j<=lines; j++)
{
if (j<=lines+1-i)
printf("%d", K++);
else
→ for alphabets "%c".
printf(" ");
}
printf("\n");
}
getch();
}

*). switch case control instruction :→

→ result for this exp. will be a Integer Value,

switch (expression)
{
→ This constant can only be a Integer Constant or character constant. It cannot be a real constant.
e.g. etc

case constant :
case constant :
case constant :
default :
}

Q). #include <stdio.h>
int main()
{
int x;
printf("Enter a number");
scanf("%d", &x);
switch (x)
{
case 70:
printf("A");
break;
case 43:
printf("B");
break;
case 0:
printf("C");
break;
case -34:
printf("D");
break;
default:
printf("Default");
}
}

(Q.). Menu driven program :=>

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int x, r, l, b, ar, n1, n2, n3;
    while(1)
    {
        printf("\n1. Area of Circle");
        printf("\n2. Area of rectangle");
        printf("\n3. Average of three numbers");
        printf("\n4. Exit");
        printf("\n\nEnter your choice");
        scanf("%d", &x);
        switch(x)
        {
            case 1:
                printf("Enter radius of a circle");
                scanf("%d", &r);
                a = 3.14 * r * r;
                printf("Area is %f", a);
                break;
            case 2:
                printf("Enter length and breadth of a rectangle");
                scanf("%d %d", &l, &b);
                ar = l * b;
                printf("Area of rectangle is %d", ar);
                break;
            case 3:
                printf("Enter three numbers");
                scanf("%d %d", &n1, &n2, &n3);
                a = (n1+n2+n3)/3.0;
                printf("Average is %f", a);
                break;
```

while(1) ← any non-zero value is always true.

Int ÷ Int → always Int
Isliye Int ÷ float kiya
Tab decimal output
dekha hai

```
case 4:
        exit(0);
default:
        printf("\n Invalid Choice");
}//end of switch.
}// end of while.
}
```

## *) What is a function?

→ Function is a block of code, which has some name for identification.

```
functionName()
{
  ——
  ——
  ——
  ——
}
```
} Function Definition.

→ Even in the smallest C program, there is atleast one function.
→ All function names must be unique.
→ One function name must be main().
→ You can define functions in any sequence.
→ No keyword is a function.
→ O.S calls main() fxn. to begin execution of program.
→ Functions are of two types:-

(1) Predefined functions → scanf(), printf(), getch(), exit(), puts(), gets(), malloc(), calloc(), etc.

(2) User defined functions →

```
main()
{
  ——
  ——
  ——
}
```
} function definition

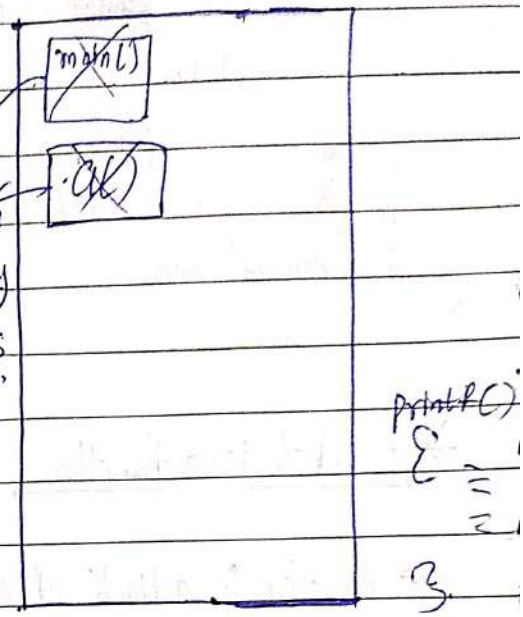
Stored in "C" library.

```
printf()
{
  ——  पहले से ही chuka hai.
  ——
}
```

Date : / /
Page :

## #). Program with multiple functions :⇒

```
main()
{
    a();
    b();
    a();
}
a()
{
    printf("Hello");
}
b()
{
    printf("Bye");
    a();
}
```




Hello Bye Hello. Hello.

## *). Ways to define a function :⇒

1. Takes Nothing, Returns Nothing (TNRN).
2. Takes Something, Returns Nothing (TSRN)
3. Takes Nothing, Returns Something (TNRS)
4. Take Something, Return Something (TSRS).

① Return Type

## TNRN ☺.

```
void add()  // Empty paranthesis means TN (Takes Nothing).
{
    int a, b, c;
    printf("Enter two numbers");
    scanf("%d %d", &a, &b);
    c = a+b;
    printf("Sum is %d", c);
}
```

Jab returns nothing hai tab.

// No return keyword means RN (Returns Nothing).

```
#include<stdio.h>
void add();   <- function declaration.
int main()
{
    add();   <- Function call.
    return 0;
}
```

Control jaaega is code pe.

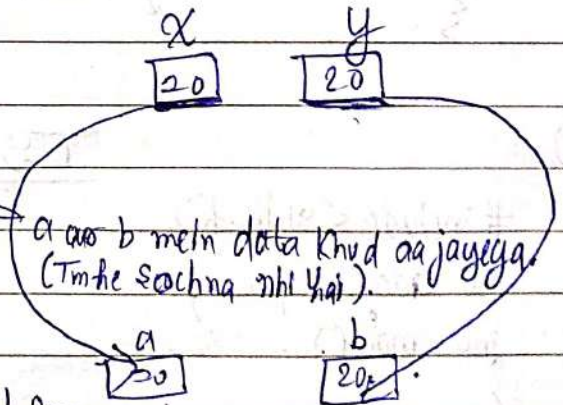
② TSRN

```
#include<stdio.h>
void add(int, int);
int main()
{
    int x, y;
    printf("Enter two nos");
    scanf("%d%d", &x, &y);
    add(x, y);
    return 0;
}
void add(int a, int b)
{
    int c;
    c = a+b;
    printf("Sum is %d", c);
}
```

Actual Arguments

// call by value

a, b → formal Argument




a aur b mein data khud aajayega. (Tmhe sochna nhi hai).

③. TNRS

```
#include <stdio.h>
int add();
int main()
{
    int s;
    s = add();
    printf("Sum is %d", s);
    return 0;
}
int add()
{
    int a, b, c;
    printf("Enter two numbers");
    scanf("%d %d", &a, &b);
    c = a+b;
    return c;
}
```

*). return ⇒
(1). it returns value.
(2). it returns control.
⇒ we cannot return more than one value using return keyword.

④. TSRS

```
#include <stdio.h>
int add(int, int);
int main()
{
    int x, y, s;
    printf("Enter two numbers");
    scanf("%d %d", &x, &y);
    s = add(x, y);
    printf("Sum is %d", s);
    return 0;
}
int add(int a, int b)
{
    int c;
    c = a+b;
    return c;
}
```

## *). What is Recursion?

→ Function calling itself is called recursive.

Most Imp. → A recursive method solves a problem by calling a copy of itself to work on a smaller problem.

→ It is important to ensure that the recursion terminates.

More Imp. → Each time the function call itself with a slightly simpler version of the original problem.

→ Recursive code is generally shorter and easier to write than iterative code.

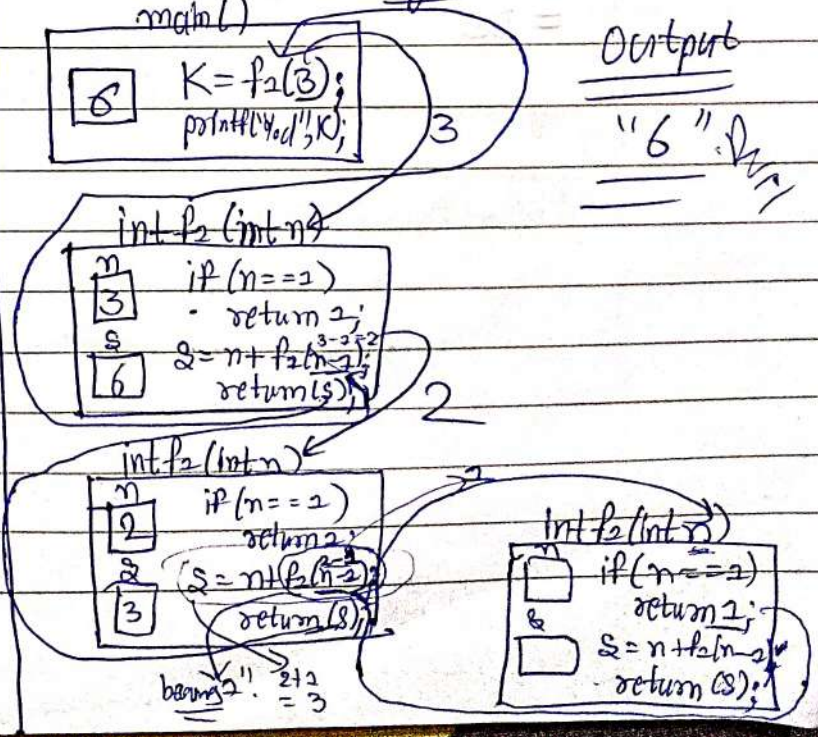
→ Solution to some problems are easier to formulate recursively.

#).

```
int main()
{
    int K;
    K = f2(3);
    printf("%d", K);
}
int f2(int n)
{
    int s;
    if(n==1)
        return 1;
    s = n + f2(n-1);
    return(s);
}
```

Recursion Tree

↓ Memory Tree / dry run.




Output "6"

→ Explanation :→ This fxn is to calculate the sum of first "n" natural numbers.

↓

$K = f_2(3)$

$\leftarrow 3 + f_2(2)$ → recursive case.

$\leftarrow 2 + f_2(1)$ ← Base case

$K = 3 + 2 + 1$

Formula ⇒ $f_2(n)$

$\leftarrow n + f_2(n-1)$.

e.g. ⇒. $K = f_2(20)$

can be replaced by

$20 + f_2(9)$

$\leftarrow 9 + f_2(8)$

$8 + f_2(7)$

$7 + f_2(6)$

$6 + f_2(5)$

$5 + f_2(4)$

$4 + f_2(3)$

$3 + f_2(2)$

$2 + f_2(1)$

$1$

Base case

$K = 20 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$.

$= 55$

## *) How to approach a Recursive Problem?

Q). Write a recursive function to calculate sum of first "n" natural numbers.

Sol)
```
int sum (int n)
{
    if (n==1)
        return(1);
    return (n+ Sum (n-1));
}
```

Steps :=> ①. Suppose koo fo fxn. banana hai wo ban chuka hai.
② . Recursive case Banao.
③. Base case Banao.

① . Sum (n) => 1+2+3+ - - - +n.
R.C ② . n + Sum (n-1) => 1+2+3+ - +n-1
B.C ③ . n==1 => return 1.

Q.) Write a recursive ~~problem~~ function to calculate factorial of n.

Sol)
```
int factorial (int n)
{
    if (n==0)
        return (1);
    return (n* factorial (n-1));
}
```

Steps :=> ② . factorial(n) => n(n-1)(n-2)..... - - - 3.2.1
R.C ② . n* fact. (n-1) => (n-1)(n-2) - - - 3.2.1
B.C ③ . n==0 => return 1.

1! = 1
0! = 1.
2! = 2X1
3! = 3X2X1

if Calculate's then Rxn is TNRS

T & RN

Q. Write a recursive function to print first "n" natural numbers.

Steps: (2). printN(n) → 1, 2, 3 --- n.

Sol":

```
void printN(int n)
{
    if (n>0)
    {
        printN(n-1);
        printf("%d", n);
    }
}
```

R.C (2). printN(n-1) → 1 2 3 --- n-1
printf("%d", n);

B.C (3). n==0 ● returns

TSRN

## Introduction to Arrays

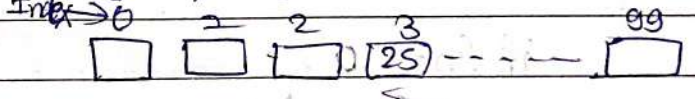
Layman → Group of variables

→ Array is a linear collection of similar elements.
→ Array is also known as subscript variable.

Q. Write a program to calculate average of 100 numbers.

Index → 0 1 2 3 --- 99 ← Subscript value.

⇒ int a[100];




$M_1, M_2, \ldots M_{100}$ ← Subscript in maths.
M[1], M[2] ← Subscript in "C".

a[3] = 25
expression

[ ] → Subscript operator
a, 3 → Operands.

i = 0 to 99
scanf("%d", &a[i]);

a[i] = 25;

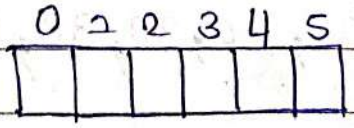
⇒
```
#include <stdio.h>
int main()
{
    float avg;
    int a[200], i, sum=0;
    printf("Enter 200 numbers");
    for (i=0; i==99; i++)
        scanf("%d", &a[i]);

    for (i=0; i==99; i++)
        sum = sum + a[i];
    avg = sum/200.0;
    printf("Average is %f", avg);

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

✱ **Array Declaration Rules**

1. `int a[];` Error — can't be empty.

2. `int a[5];`
   - Natural Number.
   - Total number of variables in array.
   - Not an index.

3. `int a[5];`




⇒ Whatever is the size of an array it always consumes memory in a sequential fashion.

⇒ local array when not initialised contains garbage values.

(4) You can initialize array during declaration.

```
int a[5] = {10, 50, 30, 70, 20};
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 10 | 50 | 30 | 70 | 20 |

(5) You cannot initialize an array during declaration more than its size.

```
int a[5] = {10, 50, 30, 70, 20, 80, 40};
```

Error

(6) You can initialize an ~~array~~ array during declaration with lesser values than the size of an array.

```
int a[5] = {10, 50};
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 10 | 50 | 0 | 0 | 0 |

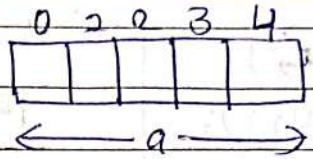
Remaining variables in array will contain "0" and not garbage value.

(7) During declaration you can leave [ ] empty only when you initialize array at the same time.

```
int a[] = {10, 50, 30, 80, 20};
```

*) **Bound Checking** → There's no boundary checking in "C" in action statements

int a[5] = {10, 20, 50, 90, 30, 60, 70}; → Error




int i, a[5];

for (i=0; i<=9; i++)
scanf ("%d", &a[i]); } → no errors but program may crash

*) **Sorting**

→ Arranging elements in some logical order is known as Sorting.

→ By default, for numbers sorting means arranging elements in ascending order.

example :→

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| given array → | 20 | 50 | 90 | 60 | 70 | 80 | 30 | 10 |
| sorted array → | 10 | 20 | 30 | 50 | 60 | 70 | 80 | 90 |

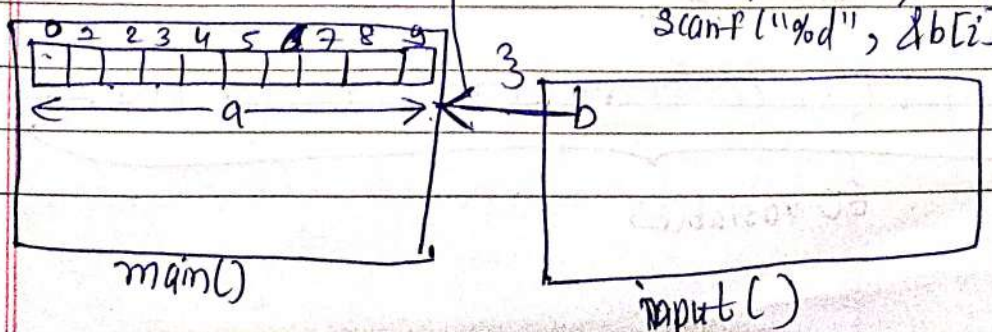
↳ write this program.

*) **Function call by passing array :→**

```
main()
{
  int a[10];
  input(a);
}
```

```
void input(int b[])
{
  int i;
  printf("Enter 10 numbers");
  for (i=0; i<=9; i++)
    scanf("%d", &b[i]);
}
```

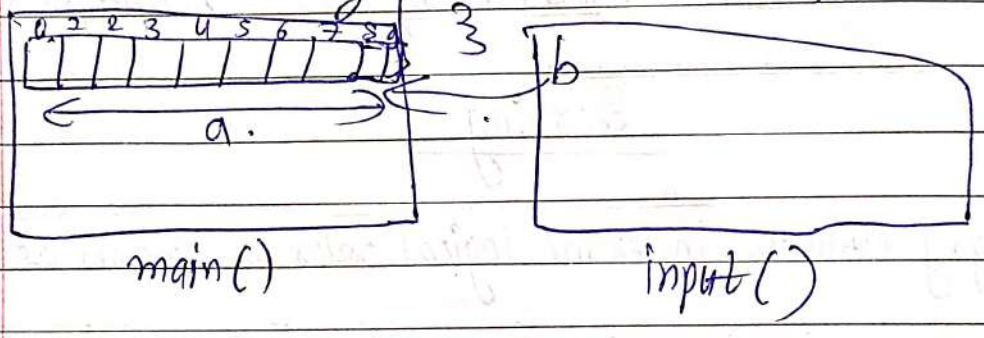
here a[], b[] are same array. Coz, iss fxn mein "a" ko access nhi kr skte isliye ek "b" ko create kiya jo same hai "a" ko.



→ isko aur ~~acche~~ bana sakte hai! (useful)

```
main()
{
    int a[6];
    input(a, 5);
}

void input(int b[], int N)
{
    int i;
    printf("Enter %d numbers", N);
    for(i=0; i<N; i++)
        scanf("%d", &b[i]);
}
```

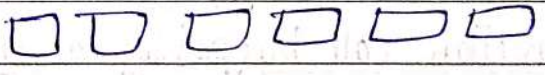
(N: size of the array; 5: size of the array)



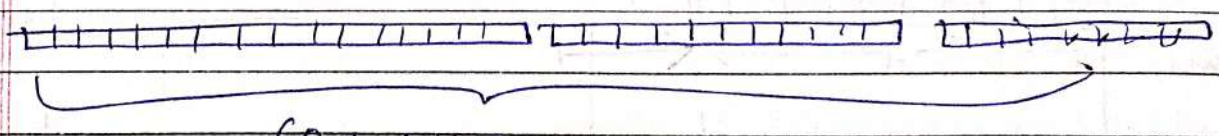

*) Two dimensional Arrays :→ (Kisi vi cheej ko measure karne ki direction ko "dimension" kehte hai.)

- int a[6]; → One-dimensional Array.


- int b[2][3]; → Two-dimensional Array. (2×3 = 6)

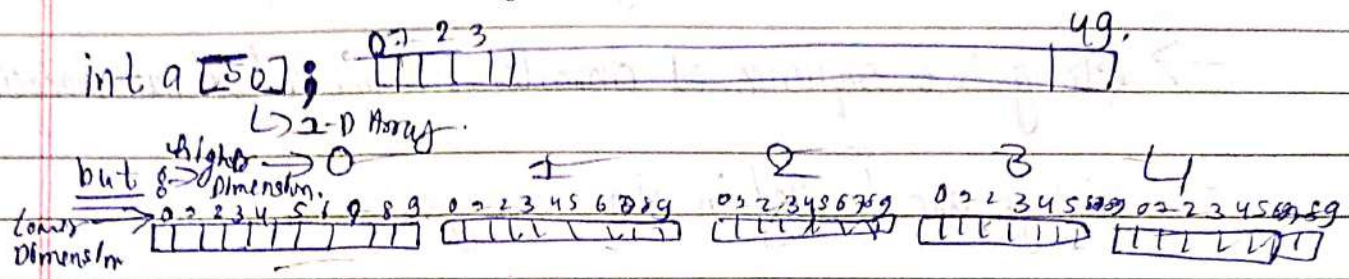


- int c[4][5][3]; → Three-dimensional Array. (4×5×3 = 60)



→ Explanation of 2-D Array :→

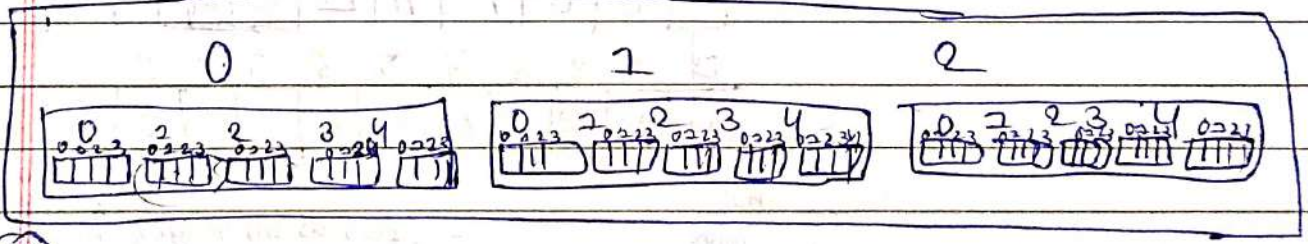
int a[5][10];
↳ 2-D Array.




Declaration :→ int a[5][10];
↑ Higher Dimension
↑ Lower Dimension.

#) Multi-Dimensional Arrays :→

int a[3][5][4];




(2-D)
Q.) int a[3][4];
int i, j;
for (i=0; i<=2; i++)
{
for (j=0; j<=3; j++)
scanf("%d", &a[i][j]);
}.

| i | j |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 0 | 2 |
| 0 | 3 |
| 1 | 0 |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 0 |
| 2 | 1 |
| 2 | 2 |
| 2 | 3 |

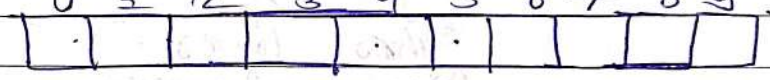
Date : / /
Page :

## *). Strings

→ String is a sequence of characters, terminated at null character.

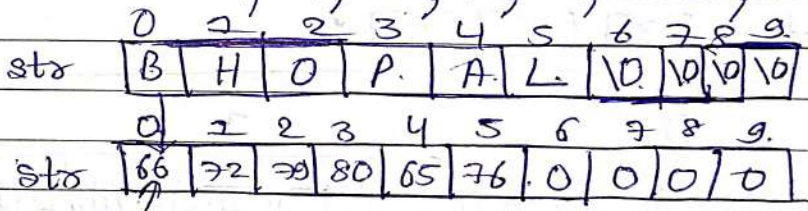
→ Strings are handled in char arrays.

```
char str[20];
```




### #). Initializing "char array" during declaration :→

```
char str[20] = {'B', 'H', 'O', 'P', 'A', 'L', '\0', '\0', '\0', '\0'};
```




ASCII value will be stored in real.

→ zero as an integer is stored

'\0' → null character → ASCII value 0 → zero

char '0' → 48.

### ##).

*) Printing String

```
int main()
{
    char str[20] = {'B','H','O','P','A','L'};
    int i;
    for(i=0; i<=5; i++)
        printf("%c", str[i]);
}
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| B | H | O | P | A | L | \0 | \0 | \0 | \0 |

Null character.

=> loop ko string ke length tak chalao.

*) Using Null character :=> Better Program.

```
int main()
{
    char str[20] = {'B','H','O','P','A','L'};
    int i;
    for(i=0; str[i]; i++)
        printf("%c", str[i]);
}
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| B | H | O | P | A | L | \0 | \0 | \0 | \0 |

But we can simply write str[i] instead of this, coz '\0' => 0 which becomes false.

str[i] != '\0'

*) "%s" :=> with this we can directly print string without using loop.

```
int main()
{
    char str[20] = {'B','H','O','P','A','L'};
    printf("%s", str);
}
```

Q.) Calcalate length of the string ⇒

⇒
```
int main()
{
                    0  1  2  3  4  5
    char str[10] = {'B','H','O','P','A','L'};
    int i;
    for (i=0; str[i]; i++);
    printf("Length is %d", i);

}
```

*) String Constant → also called string literal

↓

Jab aap → " " mein kch likhte hai

double quotes

E.g ⇒ "BHOPAL"

→
```
#include <stdio.h>
int main()
{
    char str[20] = "BHOPAL";
    printf("%s", str);
    printf("\n");
    return 0;
}
```

*) Taking inputs from user :=>

- scanf() :=>

→ scanf is not capable to input multiword strings.

⇒ because space, tab, new line characters are delimiters. (data ka end point.)

⇒ we will not use scanf for string input.

- gets() :=>

→ gets() is capable to input multiword string.

→ Program :=>

```
#include <stdio.h>
int main()
{
    char str[30];
    printf("Enter your name: ");
    gets(str);
    printf("%s", str);
    printf("\n");
    return 0;
}
```

→ gets() :=> It is deprecated from latest version "C", and it is not legit to use and it crashes the program.

Date : / /
Page :

*) fgets() → we will use this in place of gets().

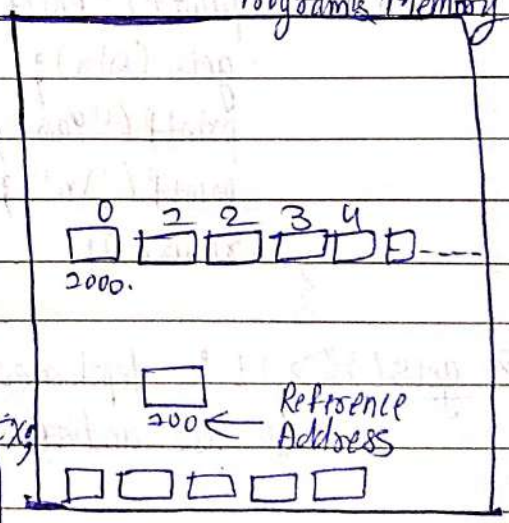
Syntax:=>
fgets(arrayname, arraysize, stdin)

→ Program :=>
```
#include <stdio.h>
int main()
{
    char str[30];
    printf("Enter your name: ");
    fgets(str, 30, stdin);
    printf("%s", str);
    printf("\n");
    return 0;
}
```

Imp. *) Memory Concept

M-I :=>
```
char str[20] = "BHOPAL";
int i;
for (i=0; str[i]; i++)
    printf("%c", str[i]);
```

M-2 :=> printf("%s", str);




str ≈ &str[0] ≈ 2000

→ array ka naam vi likhna address hota hai.

*). string.h — String Functions

- int strlen(char *); ✓ → l = strlen(str);
- char* strlwr(char*); X
- char* strupr(char*); X
- char* strrev(char*); X
- char* strcpy(char*, char*); ✓
- char* strcat(char*, char*); ✓
- int strcmp(char*, char*); ✓

(address likhna hai jaha '*' laga hai) (address.)

GH gcc compiler :o:o:> (✓) → ye hai (X) → ye hat gya hai.

returns values :=> -2 → dictionary order → means Alphabetical order.
0 → string same hai
2 → opposite of dictionary order.

*). Function Call by Passing String

→ Syntax :=> char str[20];
fgets(str, 20, stdin);
f2(str);

void f2(char str[])
{
}

str [ ] ← &

*). Handling Multiple String

→ char S[4][20] = {"BHOPAL", "PUNE", "BENGALURU", "JAIPUR"};

| | 0 1 2 3 4 5 6 7 8 9 |
|---|---|
| 0 | B H O P A L \0 |
| 1 | P U N E \0 |
| 2 | B E N G A L U R U \0 |
| 3 | J A I P U R \0 |

# *) Pointers :->

## -> Introduction :->

```
int x = 5;
printf("%d", x);     output -> 5
printf("%d", &x);    output -> 3000
printf("%d", *&x);   output -> 5
```

x = 5 (3000) <- address or Reference.
↓
always whole numbers

=> *&x ≈ x

## -> "&" :->

-> address of operator.
-> Referencing operator.
-> unary operator. -> Requires only one operand (here: "x";)
-> & — <- variable

-> variable -> & (left hai) -> address (data hai)

## -> "*" :->

-> Indirection operator.
-> Dereferencing operator.
-> unary operator.
-> * — <- address.

-> address (left hai) -> * -> variable (data hai)

## -> e.g :->

```
int x = 5;
&x = 7;
```

x = 5 (3000)

Error

@ => &x is just a way to represent address of variable x. &x is treated as a constant. &x is not a variable.

Constant = ...
left side of assignment -> error

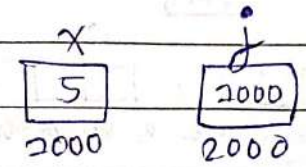
```
char str[30];
str = "BHOPAL";
```

error -> str = "BHOPAL"; correct
address

Correct way :→

strcpy (str, "BHOPAL");

next → int x = 5;




int *j; → to differentiate "j" is value from ordinary variable as we wont to store address of x. isliye "*" laga diye.
↳ ye batata hai ki "j" ke andar wala cheej address hoga aur wo special variable hoga.

j = &x;

printf("%d %d %d", j, &x, x);
output → 2000 2000 5

printf("%d %d %d", *j, *&x, &j);
5. 5 2000

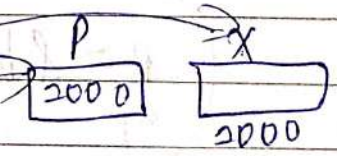
*j ≈ x

→ "j" is a pointer variable.

*) What is Pointer ? :→

→ A pointer is a variable which contains address of another variable.

int *p;
p = &x;




"p" points to x.

*p ≈ x

p Vs *p

Date : / /
Page :

4bytes ← a → variable bytes.

⇒ int a, *p;

2000 2001 2002 2003

b

char b, *q; → 2bytes

2000

c

double c, *r;

2000 3001 3002 3003 3004 3005 3006 3007.

p

1000

q

r

P = &a;
*P = 4;

8bytes

Base address

&a → 2000
&b → 2000
&c → 8000

8bytes

always for Pointer Variable. For 64bits machine, 32 bits पे 4bytes
16 bits पे 2bytes.

→ Ordinary variable का size उसके datatype पर depend करता है।

→ Pointer variable का size उसके datatype पर depend नहीं करता है।

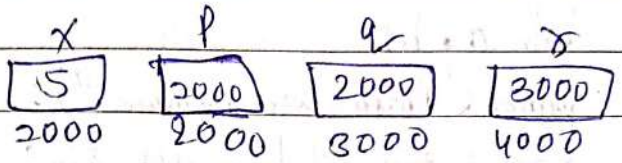
→ Pointer variable always contains Base address.

p = &b;
*p = 5; → wrong.

→ q = &b;
r = &c;

## *). Extended Concept of Pointers :⇒

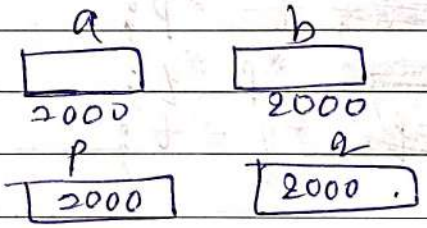
```
int x=5;
int *p, **q, ***r;
p = &x;
q = &p;
r = &q;
```




(** → Shows level of pointer.)

## *). Pointers Arithmetic :⇒

```
int a, b, *p, *q;
p = &a;
q = &b;
```




p+q
p*q
p/q
p*5
p/4
} error

p+2
p-3
(q-p)
} ✓ ⇒ Karskte ho.

dataType same hona chahiye.

p-2 → 2×4=8 = 992
p-3 → 3×4 = 988

q-p = 250 ⇒ variables

bytes ← 1000 → 4

↳ It is not normal arithmetic addition

e.g. :⇒ p+1 ⇒ 1004
Base address → 4 bytes → next variable ka Base address aayega.

p+2 → 2×4=8 ⇒ 1008
p+5 ⇒ 1020
5×4=20 bytes

Fxn. declaration

void swap (int *, int *); ⇒ make a swapping fxn.

Q.) int main ()
{
int a, b;
printf ("Enter two numbers");
scanf ("%d %d", &a, &b);
// swapping. ⇒ calling swap (&a, &b);
printf ("a= %d b= %d", a, b);
}

a: 10, 2000 ; b: 20, 2000

swapping.

a: 20, 2000 ; b: 20, 2000

call by Reference

Sol" ⇒

void swap (int *p, int *q)
{
int t;
t = *p;
*p = *q;
*q = t;
}

p: 2000 ; q: 2000

t: 10.

## Pointers and Arrays ⇒

int a[10];
int *p;
p = &a[0];

p: 2000

0 1 2 3 4 5 6 7 8 9

2000 2004 2008 2012 2016 2020 2024 2028 2032 2036.

*(p+0) ≈ a[0]
*(p+1) ≈ a[1]
*(p+i) ≈ a[i].

a[3]
solved aise hota hai ⇒ *(a+3)
*(2000+3)
*(2012).

p[3]
*(p+3)
*(2000+3)
*(2012)

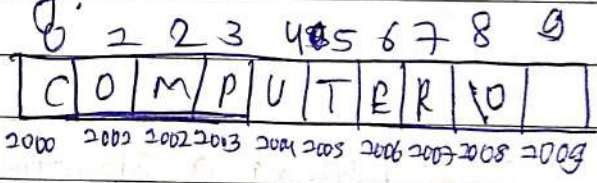
⇒ p[3] ≈ a[3]

⇒ "a" ⇒ mein kch assign nhi kr sktte.
⇒ "p" ⇒ mein kch assign kr sktte hai.
variable

## *) Pointers and Strings :⇒

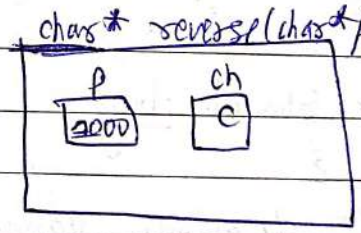
```
int l;
char str[20] = "BHOPAL";
l = length(str);

int length (char *p)
{
    int i;
    for (i=0; p[i]; i++);
    return i;
}
```

Q). Write a function to reverse a string?




Sol:) ~~char* reverse(char *p)~~
~~main()~~

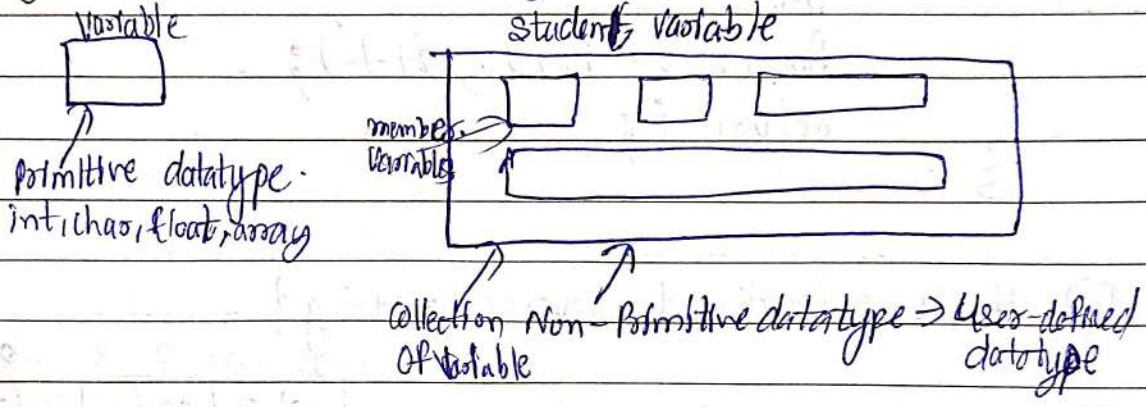
```
char* reverse(char*);
main()
{
  clrscr();
  printf("%s", reverse("COMPUTER"));
  ~~printf(~~
  getch();
}

Char* reverse(char *p)
{
  int l, i;
  char ch;
  for (l=0; *(p+l) != '\0'; l++);
  for (i=0; i<l/2; i++)
  {
    ch = *(p+i);
    *(p+i) = *(p+l-1-i);
    *(p+l-1-i) = ch;
  }
  return (p);
}
```



*) # Structure in C Language

Q) What is Structure?

Ans) → Structure is a way to group variables.
→ Structure is a collection of dissimilar elements.
→ Defining Structure means creating new datatype.




→ Defining a Structure :=>

name of user made datatype.

```
Struct tag
{
  // member variable declarations here.
};
```

Q). Data ki information store Karni hai :=> (Student.)

Soln=>

```
struct date
{
  int d, m, y;
};
```

```
struct student
{
  int rollno;
  char name[20];
  int age;
};
```

→ No memory is consumed for definition of structure.

→ Whenever we use user defined datatype to create variables we should always write "struct" keyword before datatype.

Syntax:→ struct date today;

struct → using kiya compulsory hai. date → datatype. today → variable

⇒ Program to understand:→

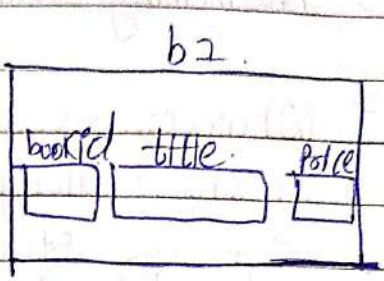
→
```
struct date
{
  int d, m, y;
};
void main()
{
  struct date today, d1;
  today.d = 26;
  today.m = 7;
  today.y = 2025;

  d1 = today;
  clrscr();
  printf("Enter today's date");
  scanf("%d/%d/%d", &d1.d, &d1.m, &d1.y);
  printf("Date: %d/%d/%d", d1.d, d1.m, d1.y);
  getch();
}
```

→ Manual Entry.

Date : / /
Page :

Ex 32 (P) structure

```
#include<stdio.h>
struct book
{
    int bookid;
    char title[20];
    float price;
};
struct book input()
{
    struct book b;
    printf("Enter bookId, title, Price");
    scanf("%d", &b.bookid);
    fflush(stdin);
    gets(b.title);
    scanf("%f", &b.price);
    return(b);
}
void display(struct book b)
{
    printf("\n %od %c %f", b.bookid, b.title, b.price);
}
void main()
{
    clrscr();
    struct book b1;
    b1 = input();
    display(b1);
    getch();
}
```




} ⇒ in new update we can use. → fgets(b.title, 20, stdin);

## *) Dynamic Memory Allocation :=>

- SMA :=> Static Memory Allocation → compile-time Allocation
- DMA :=> Dynamic Memory Allocation → runtime Allocation

↳ Isse banne wala variable ka naam nhi hota, sirf address hota hai.

=> Explanation of SMA :=>

```
main()
{
    int a;      // SMA
    float b;    // SMA
    int x[5];   // SMA
    ...
}
```

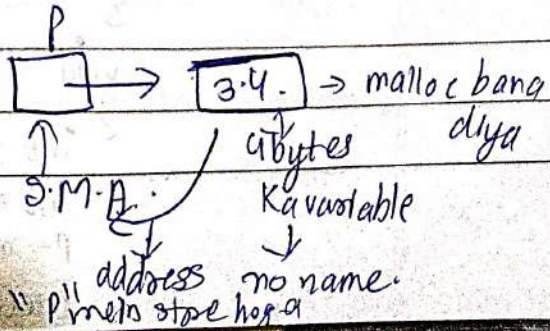
## #). DMA :=> Explanation

- malloc()
- calloc()
- realloc()
- free()

## => malloc() :=> Syntax or Program :=>

```
=>main()
{
    float *p;
    if()
    p = (float*)malloc(4);
    *p = 3.4;
}
```

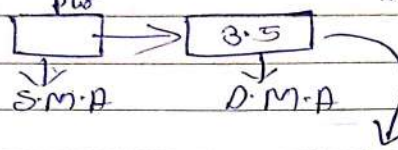
typecasting



## malloc() V/S calloc() :=>

**malloc()** → Block banta hai

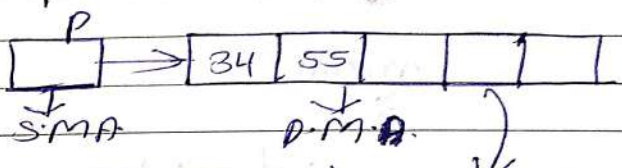
```
float *ptr;
ptr = malloc(4);
*ptr = 3.5;
```




Contains Garbage value initially

**calloc()** → Array banata hai

```
int *p;
p = calloc(5,2);
```




```
*(p+0) = 34;
*(p+1) = 55;
```

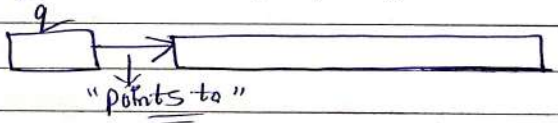
Contains "0" in each block.

## realloc() :=> For resizing the memory block made by malloc() or calloc().

**Syntax :=>** `void* realloc(void *block, int size);`

eg =>

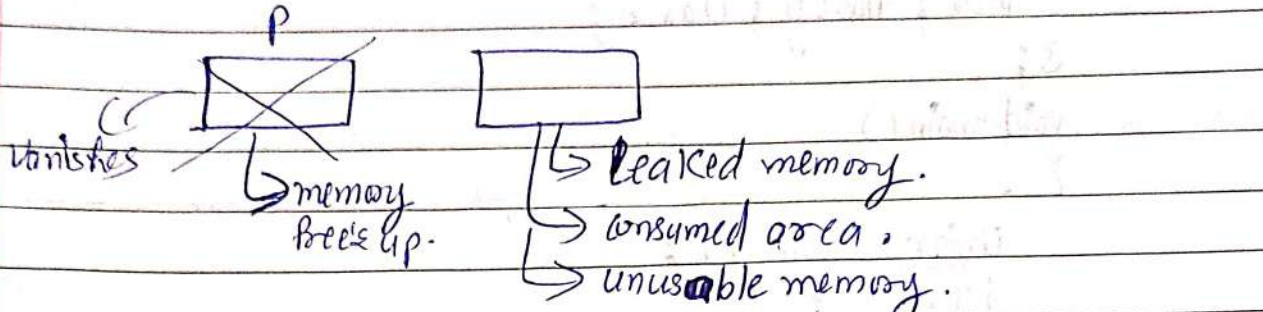
```
double *q;
q = realloc(ptr, 8);
```




→ **Program :=>**

```
main()
{

}
void fun()
{
  int x; // local variable SMA.
  int *p;
  p = malloc(2);
  free(p); => used to free up memory of DMA variables created by malloc() or calloc()
}
```

#. free() :- Used to free DMA variables memory ~~use~~ created by malloc() or calloc().




## Union in C :-

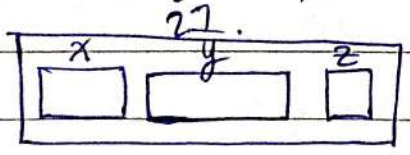
→ Union is similar to structure, except it allows you to define variables that share storage space.

→ Defining union means creating new datatype.

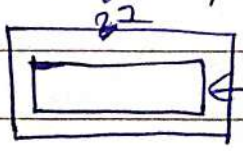
Q.) → What is union?

⇒

```
Struct item
{
  int x;
  Float y;
  Char z;
};
struct item i1;
```




```
union item
{
  int x;
  float y;
  char z;
};
union item i1;
```




→ Union is memory efficient but it cannot be used every where.

Q.). union Item
{
int x ; float y ; char z ;
};
void main()
{
Union Item i1 ;
i1.x = 5;
printf ("\n x= %d", i1.x);
i1.y = 3.5 ;
printf ("\n y= %f", i1.y);
i1.z = 'a';
printf ("\n z= %c", i1.z);
getch();
}

Q.) How to access ?

⇒ Union members are accessed in the same manner as we access structure member.

note ⇒ Union ki jyada jarurat low level language mei hoti hai jabki structure High level language mei padti hai.

## *) Enumerators in C language :->

#) What is Enumerator ?

-> It gives an opportunity to invent own datatype and Define what values the variable of this data type can take.

eg :-> enum month
{
jan, feb, mar, apr, may, jun, jul, aug, sept, oct, nov, dec.
};
main()
{
enum month m2, m2, m3;
}.

-> Internally, compiler treats the enumerators as integers.

-> Each value on the list of permissible values corresponds to an integer, starting with "0", in the example, jan is stored as "0", feb is stored as "1", ....., dec is stored as "11".

-> We can initialize enumerators with different integer values

-> enum mont.
{
jan = 1, feb, mar, apr, may, jun, jul, aug, sept, oct, nov, dec.
};
main() {
enum month m2, m2, m3;
}

Date : / /
Page :

→ e.g. :→
```
enum boolean
{
  false, true.
};
```

→ You can write any program in "C" language without the help of enumerators but, enumerations helps in writing clear codes and simplify programming.

→ Program :→

⇒
```
enum boolean
{
  false, true
};
enum boolean isEven(int x)
{
  if (x%2==0)
    return (true);
  else.
    return (false);
}
void main().
{
  int n;
  closes();
  printf("Enter a number");
  scanf("%d", &n);
  result = isEven(n);
```
→ enum boolean result;

```
if (result == true)
    printf ("Even number");
else
    printf ("Odd number");
getch();
}
```

*) typedef in "C" language :=>

Q:) What is typedef?

Soln =>

=> typedef is a Keyword.

=> We can use typedef to give new name to a data type.

Syntax :=>

```
typedef int LENGTH;
```

=> Now you can use LENGTH as a data type which is just same as int.

```
LENGTH x, y;
```

=> By convention, uppercase letters are used for these definitions to remind the user that the type name is really a symbolic abbreviation, but you can use lowercase.

```
typedef int length;
```

Date : / /
Page :

## Typedef in a Structure?

→
```
typedef struct student
{
    int rollno;
    char name[20];
    int age;
} STUDENT;
void main()
{
    STUDENT s1;
    s1.rollno=34;
    strcpy(s1.name, "AKSHAT");
    s1.age=20;
}
```

## What is Pre-processor?

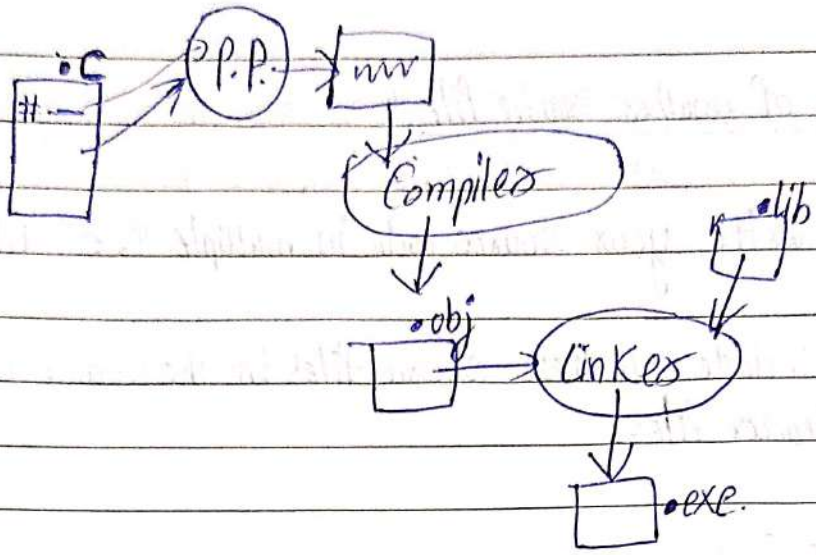
→ Pre-processor is a program which performs before the compilation.

→ Pre-processor only notices "#" started statements.

→ "#" is called pre-processor directive.

→ Each preprocessing directive must be on its own line.

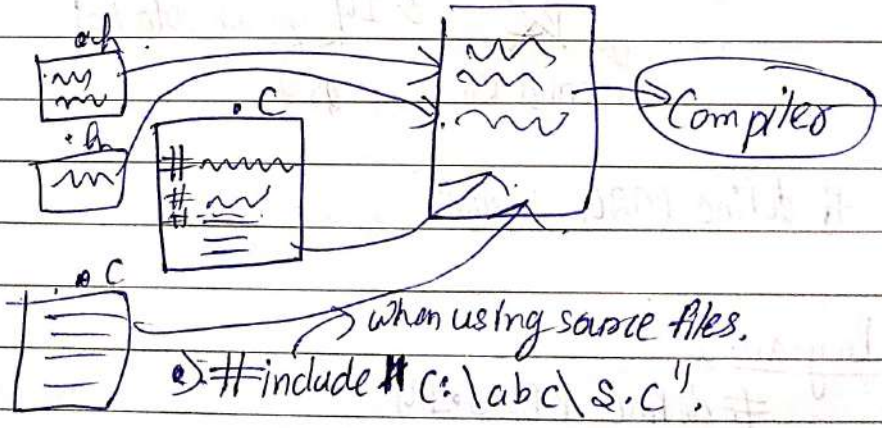
→ The word after "#" is called pre-processor command.




## *) #include :->

-> include is one of the most popular pre-processor command.
to include.
-> It can be used any file content to your sourse file.

- #include <file_name>
- #include "file_name".




-> Remember :->

- #include <C:\myprog\clang\list.h> is wrong.
- #include "C:\myprog\clang\list.h" is correct.

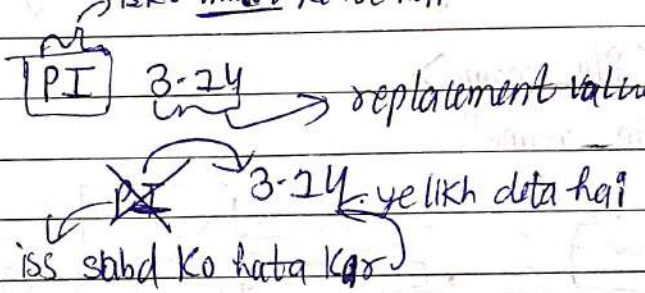
Date : / /
Page :

## #) → Inclusion of another Source file :→

- You can write your source code in multiple ".c" files.
- You can include all these source files in the same way as you include header files.

## *) #define :→

- The #define directive defines an identifier and a character sequence (a set of characters) that will be substituted for the identifier each time it is encountered in the source file.




```
#define MSG "hello"
```

→ Program :→

```
#define PI 3.14
int main()
{
    int r;
    float a;
    printf("Enter radius of a circle");
    scanf("%d", &r);
    a = PI*r*r;
    printf("Area of circle is %f", a);
} getch();
```

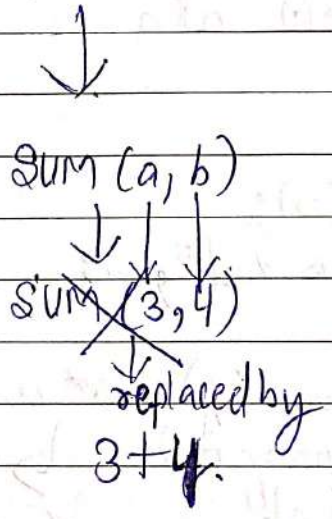
#3. Defining Macro like a function :=>

e.g. :=> ① #define ABS(a) (a)<0?-(a):(a).

```
main()
{
printf("abs of -2 & 2: %d %d", ABS(-2), ABS(2));
}
```

e.g. :=> ② #define SUM(a,b) a+b

```
int main()
{
    printf("Sum of 3 and 4 is %d", SUM(3,4));
    getch();
}
```




a+b
3+4

e.g. :=> ③ #define PRODUCT(a,b) a*b

```
int main()
{
    printf("Product of 3+2 and 4-6 is %d", PRODUCT(3+2, 4-6));
    getch();
}
```

PRODUCT(a,b) a*b.

Replaced by PP compiler ⇒ PRODUCT(3+2, 4-6)

a=5 ✗
a=3+2 ✓

⇒ 3+2 * 4-6
⇒ 3+8-6 ⇒ 11-6
⇒ ⑤.

Date : / /
Page :

e.g. Q:> #define PRODUCT(a,b) (a)*(b)

```
int main()
{
    printf("Product of 3+2 and 4-6 %d", PRODUCT(3+2,4-6));
    getch();
}
```

PRODUCT(a,b) (a)*(b)

PRODUCT (3+4, 4-6) → (3+2) * (4-6)

⇒ (5) * (-2)

⇒ −10.

Q. #define square(a) a*a.

```
main()
{
    int s = square(5);
    printf("square is %d", s);
}
```

Output ⇒ square(a) a*a.

square(5) → 5*5

replaced by

5*5

s = 25.

*) #undef :> #define ke define kiye huve macro ko undefine karta hai.

→ It is used to undefine Macros.

→ Example :> #define CLOSE 0

---

---

#undef CLOSE.

---

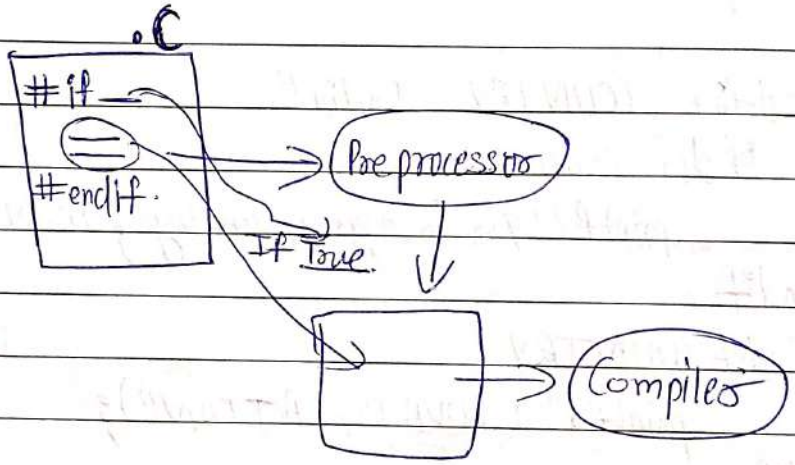
---

## *). Pre-processor Commands :=>

- #if, #else, #elif, #endif
- #ifdef, #ifndef
- ##

## #) #if, #else, #elif, #endif :=>

=> As a file is being compiled, you can use these commands to cause certain lines of code to be included or not included (for compilation).




e.g :=> #define COUNTRY India
#if COUNTRY == pakistan
printf ("Pakistani Rupee");
#elif COUNTRY == bangladesh.
printf ("Taka");
#elif COUNTRY == Nepal.
printf ("Nepali Rupees");
#else
printf ("Indian Rupees");
#endif

Date : / /
Page :

## #) #ifdef, #ifndef :=>

→ #ifdef macro
→ If the macro has been defined by a #define statement, then the code immediately following the command will be compiled.

→ #ifndef
→ If the macro has not been defined by a #define statement, then the code immediately following the command will be compiled.

eg. :=> #define COUNTRY "India"
#ifdef COUNTRY
printf("%s is a great country", COUNTRY);
#endif
#ifndef COUNTRY
printf("I LOVE MY NATION");
#endif.

## #) ## :=>

→ The ## operator is used with the #define macro.
→ Using ## concatenates what's before the ## with what's after it.
→ #define ACTION(a,b) a##b+a*b.
main()
{
printf("%d", ACTION(3,4)); ⇒ 34 + 3*4
}
(34) + 3*4.
a b
will be replaced
⇒ (46) Ans

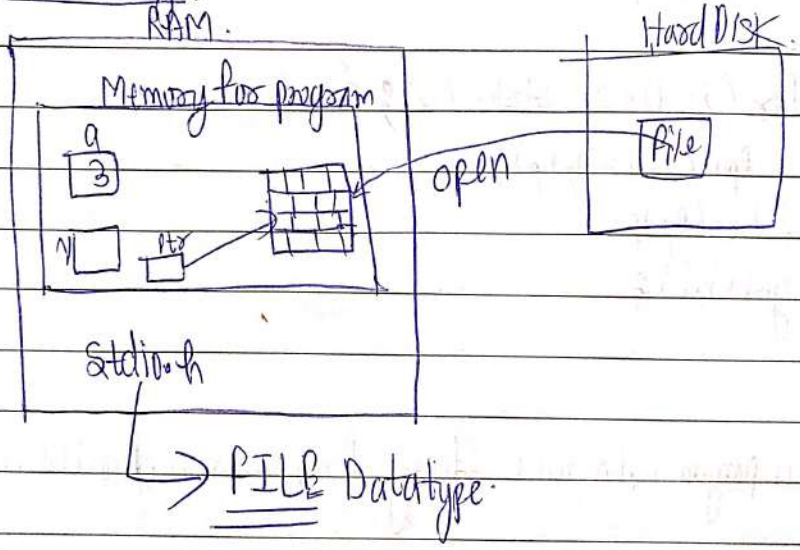
## *) What is file handling?

- Data
- Data Persistence
- Files
- File Handling.

→ In Simple Words,

- If same data is to be processed again at some later stage, again we have to enter it.

→ file handling :⇒



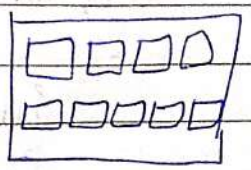

→ FILE Datatype.

# FILE :⇒ It is a datatype.

→ FILE is a structure (non primitive datatype)

→ typedef struct.
{

```
short level;
unsigned flags;
char fd;
unsigned char hold;
short bsize;
unsigned char *buffer;
unsigned char *curp;
unsigned istemp;
short token;
} FILE;
```



*). Writing in a file :->

Q.). Write a program to write "Hello students" in a file.

Sol^n=>
```
#include<stdio.h>
int main()
{   int i;
    char s[] = "Hello students";
    FILE *fp;
fp=fopen("f1.txt","w");
   if (fp == NULL) {
        printf("File cannot open");
        exit(0);
    }
    for (i=0; i<strlen(s); i++)
      fputc(s[i],fp);
    fclose(fp);
    getch();
}
```

Q.). Write a program to take string from user and write it into a file.

Sol^n=>
```
#include<stdio.h>
int main()
{
    int i;
    FILE *fp;
    char s[200];
    fp= fopen("f2.txt","w");
```

```
if (fp == NULL) {
    printf ("File cannot open");
    exit(1);
}
```
"1" indicates that, it is an abnormal terminate.
→ Matlb kisi condition mein hmne program ko end karaya hai.

```
printf ("Enter a string");
gets(s);
for (i=0; i<strlen(s); i++)
  fputc(s[i], fp);
getch();
fclose(fp);
getch();
}
```

## *) File Opening Modes :->

### #) fopen() :->

- `fp = fopen ("filename", "mode");`

### => Program :->

```
#include <stdio.h>
int main()
{
    FILE *fp;
    fp = fopen("myfile.dat", "r");
    if (fp == NULL)
        printf("File Not Found");
}
```

| Mode | Meaning | Description |
|---|---|---|
| r | Read | Only reading possible. Not create file if not exist. |
| w | write | Only writing possible. Create file if not exist otherwise erase the old content of file and open as a blank file. |
| a | Append | Only writing possible. Create file if not exist, otherwise open file and write from the end of file (do not erase the old content). |
| r+ | Reading + writing | R & W possible. Create file if not exist. Overwriting existing data. Used for modifying content. |
| w+ | Reading + writing | R & W possible. Create file if not exist. Erase old content. |
| a+ | Reading + Appending | R & W possible. Create file if not exist. Append content at the end of file. |

## Reading from a file :=>

→ Extracting data from a file to our program variables.
→ This will not remove data from the file.

→ How to start :=>

```
main()
{
  char ch;
  File *fp;
  fp = fopen("somefile.txt", "r");
  ---
}
```

Q). Write a program to read content from a file and display on the screen.

Sol^n =>

```
#include <stdio.h>
main()
{
    char ch;
    FILE *fp;
    fp = fopen("f2.txt", "r");
    if (fp == NULL) {
        printf("File Not Found");
        exit(0);
    }
    ch = fgetc(fp) {
    while (!feof(fp)) {
        printf("%c", ch);
        ch = fgetc(fp);
    }
    fclose(fp);
}
```

*) Reading from a file using fgets() :->

-> fgets() is a function to read string from a file.
-> fgets(str, n, fp);
-> fgets returns a NULL value when it reads EOF.
   ↳ End of file.

Q) Write a program to read content from a file and display on the screen. Use fgets() to read string from the file.

Sol:⇒
```
#include <stdio.h>
main()
{
    char str[20];
    FILE *fp;
    fp = fopen("f1.txt", "r");
    if (fp == NULL).{
        printf("File Not found");
        exit(1);
    }                    → Array se kam length rakho
while(fgets(str, 9, fp) != NULL){
    printf("%s", str);
}
    fclose(fp);
}
```

*) Writing in a file using fputs() :⇒

#) fputs() :⇒

→ fputs() is a function declared in stdio.h header file.
→ fputs(str, fp);

Q) Write a program to write a string to a file. Use fputs() to write content to the file.

Sol"⇒
```
#include <stdio.h>
int main()
{
    char str[20];
    FILE *fp;
    fp = fopen ("f1.txt", "a");
    printf ("Enter your name");
    gets (str);
    fputs (str, fp);
    fclose (fp);
}
```

*) Writing in a file using fwrite() function :⇒

#) fwrite() :⇒

⇒ fwrite() function is used to write content to the file in binary mode.

⇒ int fwrite (void *Buffer, int Size, int Count, FILE *ptr);

#) Program :⇒

⇒
```
#include <stdio.h>
int struct book
{
    int bookid;
    char title[20];
    float price;
};
```

```
void main()
{
    struct book b1;
    FILE *fp;
    fp=fopen("my-file.dat","wb");
    printf("Enter bookid, title and price");
    scanf("%d",&b1.bookid);
    fflush(stdin);
    gets(b1.title);
    scanf("%f",&b1.price);
    fwrite(&b1, sizeof(b1), 1, fp);
    fclose(fp);
}
```

## *) Reading from a file using fread() :=>

### #) fread() :=>

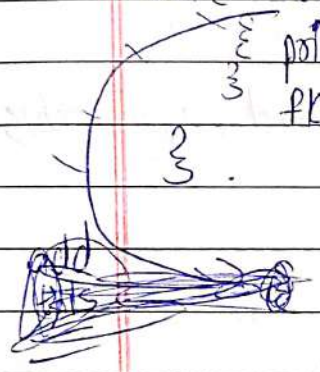
→ fread() function is used to read content from file in binary mode.

→ int fread(void *Buffer, int size, int count, FILE *ptr);

### #) Program :=>

```
#include <stdio.h>
struct book
{
    int bookid;
    char title[20];
    float price;
};
```

```
void main ()
{
    struct book b1;
    FILE *fp;
    fp = fopen ("myfile.dat", "rb");
    if (fp == NULL) {
        printf ("File Not Found");
        exit(1);
    }
while (fread (&b1, sizeof(b1), 1, fp));
    { printf ("%d %s %f", b1.bookid, b1.title, b1.price);
    fclose(fp);
}
```



~~printf("%d %s %f\n", b1.bookid,~~

*) Writing in a file using fprintf() function :=>

#) fprintf() :=>

=> fprintf() function is used to write formatted output to the specified stream.

=> int fprintf (FILE *stream, const char *format [, argument, ...]);

eg

=> fprintf (fp, "\nSum of %d and %d is %d", a, b, c);

Q. Write a program to write content to a file. Use fprintf() to write content to the file.

Soln ⇒
```
#include<stdio.h>
int main()
{
    FILE *fp;
    int a,b;
    fp = fopen("f2.txt","w");
    printf("Enter two numbers");
    scanf("%d %d",&a,&b);
    fprintf(fp,"sum of %d and %d is %d",a,b,a+b);
    fclose(fp);
}
```

*) Reading from a file using fscanf() function :⇒

#) fscanf() :⇒

→ fscanf() function is used to read formatted content from file.

→ int fscanf(FILE *stream, const char *format,...);

→ Reads data from the stream and stores them according to the parameter format into the locations pointed by the additional arguments.

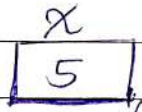
Q.) Write a program to read content from a file and display on the screen. Use fscanf() to read content from the file.

Sol$^{n}$ ⇒

```
#include <stdio.h>
main()
{
    FILE *fp;
    int a, b, c;
    fp = fopen("f2.txt", "r");
    fscanf(fp, "%d %d %d", &a, &b, &c);
    printf("a= %d .b= %d c= %d", a, b, c);
    fclose(fp);
}
```

## * Storage Classes :⇒

```
int x = 5;
```




↳ 4 bytes

## # Characteristics of a variable :⇒

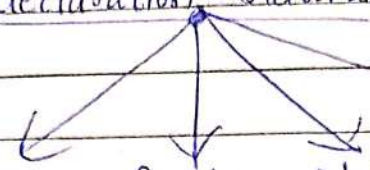
⇒ Three properties of Variables :⇒
- ⇒ Name of variable → "x"
- ⇒ Size of memory block → "4 bytes"
- ⇒ Type of content ⇒ "int".

## # Other properties of variable :⇒

| | |
|---|---|
| ⇒ Default Value | ⇒ Scope |
| ⇒ Storage | ⇒ Life. |

Date : / /
Page :

→ Koi vi declaration statement 4 types ka hota ⁰/₀ →



Auto-matic Storage class

Register Storage Class

Static Storage class

External Storage class → yhi sb storage class hota hai.

By default yhi hota

e.g. int x = 5;

yha agar kch nhi kal toh by default auto hoga.

## ⇒

| Storage Class | Keyword | Default Value | Storage | Scope | Life. |
|---|---|---|---|---|---|
| Automatic | auto | Garbage | RAM | Limited to the block in which it is declared. | Till the execution of the block in which it is declared. |
| Register | register | Garbage | register | Same | Same. |
| Static | static | 0 (zero) | RAM | Same | Till the end of Program |
| External | extern | 0 (zero) | RAM | Global | Same. |

#) Program :- for auto :-

```
main()
{
    int x = 5;
    printf("%d\n", x);
    {
        int x = 2;
        printf("%d\n", x);
    }
    printf("%d\n", x);
}
```

(2) for register :-

```
main()
{
    register int x = 4;
    int y;
    y = x++;
    x--;
    y = x+5;
}
```

(3) for static :-

```
void f1();
main()
{
    f1();
    f1();
}
void f1()
{
    static int i;
    i++
    printf("i = %d\n", i);
}
```

(4.) for external :⇒ (extern Keyword)

```
=> int x;
main();
{
    extern int x;
    printf("x=%d", x);
    f1();
    printf("x=%d", x);
}
Void f1()
{
    x++;
    printf("x=%d", x);
}
```

*) Bit fields :⇒

→ Bit fields are used to consume memory efficiently when we know that the value of a field or group of fields will never exceed a limit or is within in a small range.

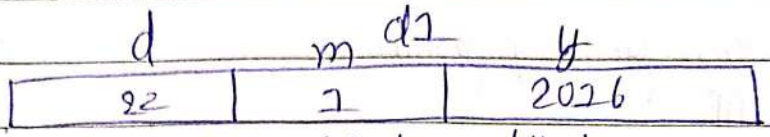
→ It can be used in structure and union.

→ Example :⇒

```
struct date
{
    unsigned int d;
    unsigned int m;
    unsigned int y;
};
void main()
{
    struct date d1 = {22, 1, 2016};
    printf("size of d1 is %d", sizeof(d1));
}
```

The variable "d1" of type 'date' takes 12 bytes on a compiler where an unsigned int takes 4 byte

→ Explanation :→ 32-bit System :→




4 byte = 32 bits  4 byte = 32 bits  4 byte = 32 bits  = 12 bytes.

31 ⇒ 11111 (5 bits)

22 ⇒ 10110 (4 bits)

→ Conclusion :→

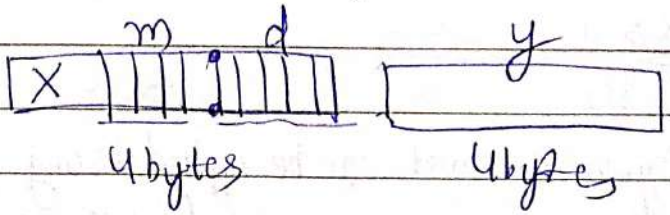
- We know that,
  → the value of "d" is always from 1 to 31.
  → value of "m" is from 1 to 12.

- We can optimize the space using bit fields.

→ modified examples using bit fields :→

```
struct date
{
    unsigned int d : 5;   → Bits
    unsigned int m : 4;
    unsigned int y;
};
struct date d1 = {22, 1, 2026};
```

The variables d1 of type 'date' takes 8 bytes on a compiler where an unsigned int takes 4 bytes.



Date : / /
Page :

*) What is command line ?

- Ways to run your program
  → Using IDE
  → By double click.
  → using command line.

*) const Keyword :=>

- const is a keyword in C language.
- const is a qualifier.
- The qualifier const can be applied to the declaration of any variable to specify that its value will not be changed.

→ example :=>

```
main ()
{
    const int x=5;
    x++;
    printf ("x= %d", x);
}
```

→ Program will show error bcoz "x" can't be incremented

- const was not there in early C, the concept is borrowed from C++.

#) Usage :=>

- The keywords const can be applied to any declaration, including those of structures, unions, enumerated types, or typedef names.

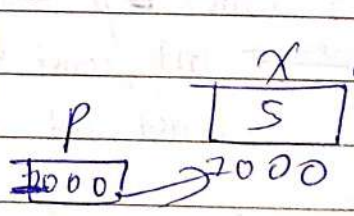
- Applying it to a declaration is called qualifying the declaration.
- const means that something is not modifiable.

→ Program :- ① main()
{
const int x = 5;
printf("x = %d", x);
x++;
printf("x = %d", x);
}

② main()
{
const int x;
printf("x = %d", x);
}


→ error.

③ main()
{
const int x = 5;
int *p;
p = &x;
printf("x = %d", x);
++(*p);
printf("x = %d", x);
}




→ no error

Power of pointer.

④
```
main()
{
    const int x=5;
    const int *p; //pointer to const.
    p=&x;
    printf("x=%d", x);
    ++(*p);
    printf("x=%d", x);
}
```

→ const pointer ⇒ mtlb (p++) → nhi krsktc the.
P = ✗ *p = ✓

→ pointer to const → (++*p) nhi kar sktc.
P = ✓ *p = ✗

→ Const pointer to const ⇒ P = ✗
*p = ✗

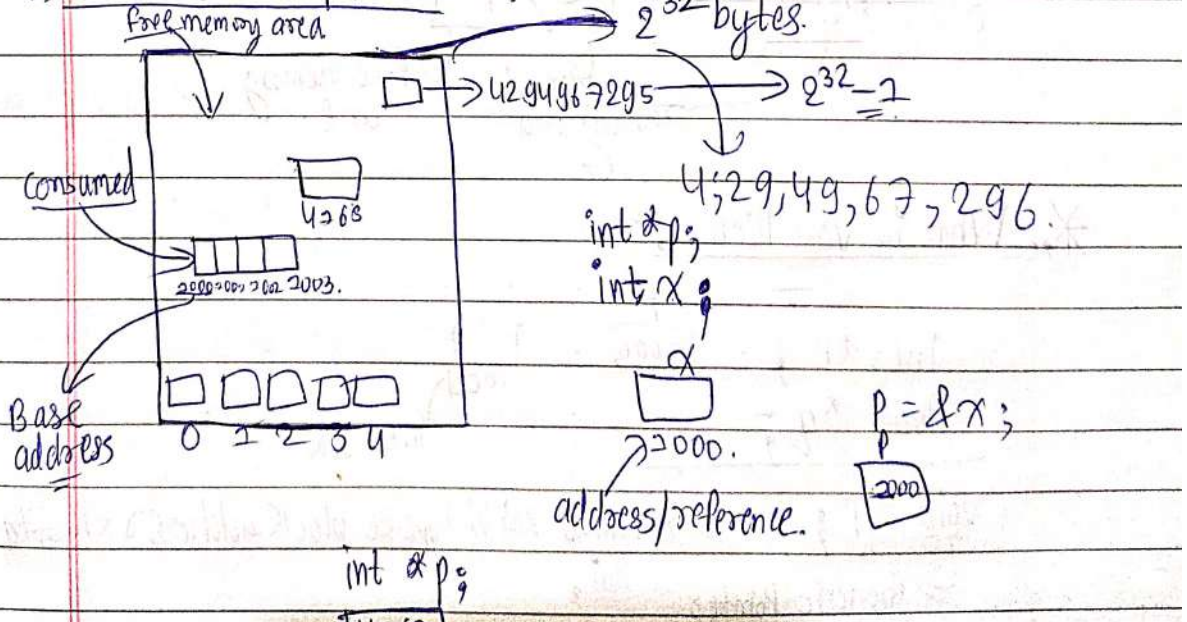
# ⇒ ~~int~~ int *const p ⇒ const pointer.
~~int const~~ int const *p ⇒ } ⇒ pointer to const.
const int *p ⇒ }

⑤
```
main()
{
    const int x=5;
    int *const p;
    p=&x;
    printf("x=%d", x);
    ++(*p);
    printf("x=%d", x);
}
```

(6).
```
main()
{
    const int x=5;
    int * const p= &x;
    printf("x=%d", x);
    ++p;
    printf("x=%d", x);
}
```

(7).
```
main()
{
    const int x=5;
    const int * const p=&x;
    printf("x=%d", x);
    ++(*p);
    printf("x=%d", x);
}
```

*) What is wild pointer? =>



→ Program :=>

```
main()
{ int x;
  int *p; //uninitialized or wild pointer.
  *p = 4;
  p = &x;
}
```

*) What is NULL Pointer ? :=>

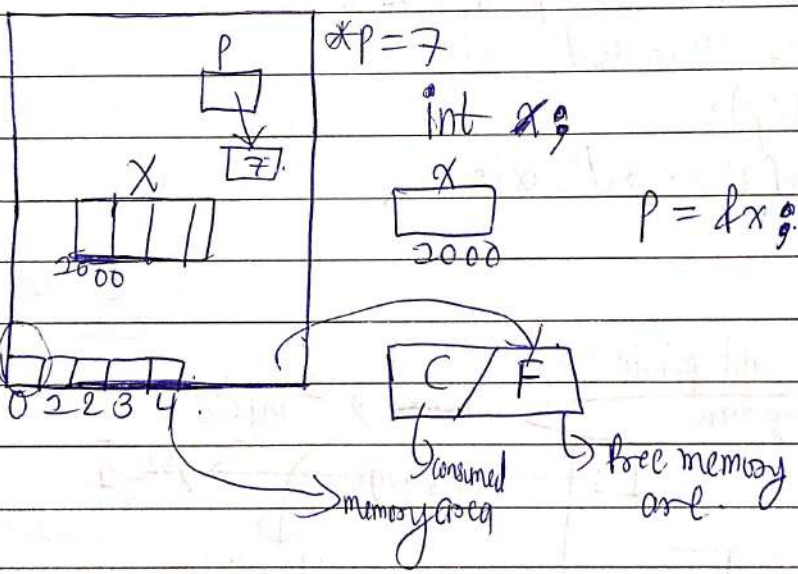
→ To #define command se banta hai

Macro.

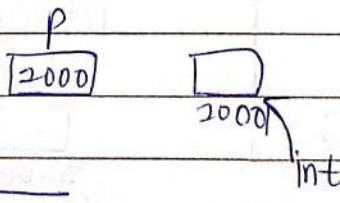
```
int *p = NULL;
```

defined in stdio.h.

```
#define NULL 0
```




*) What is void Pointer ? :=>



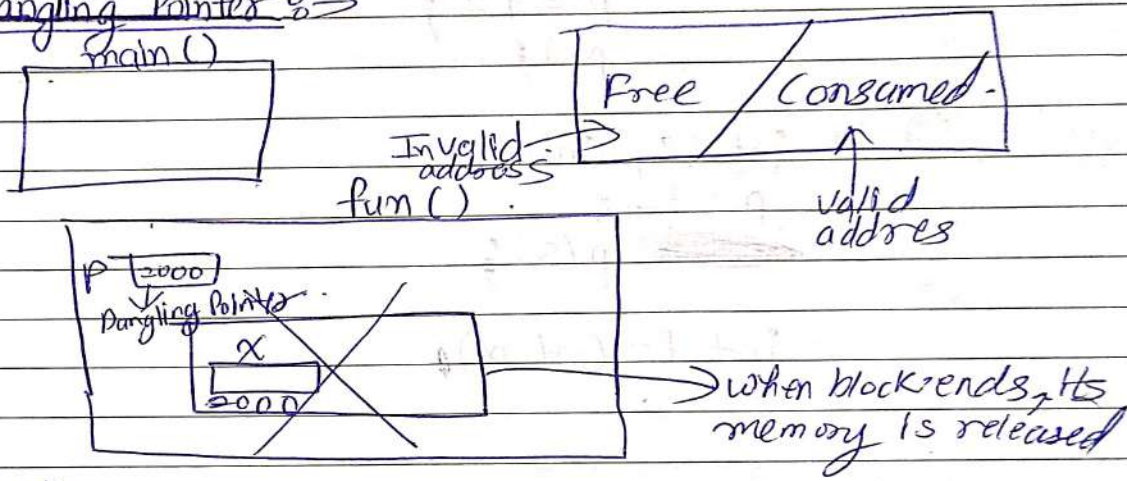

void *p; → Ye Pointer kisi bhi type ke block address rakh skta hai.

↳ Generic Pointer

```
void *p;
int x
p = &x;
*(int*)p = 10;
  ↳ Typecasting

float y;
p = &y;
*(float*)p = 4.7;
```



## What is Dangling Pointer ?=>




```
=> void fun(void);
   main();
   {
       fun();
   }
   void fun()
   {
       int *p;
       {
           int x;
           p = &x;
           ...
       }
       p = NULL;
   }
```

3.

Date : / /
Page :

*) What is function Pointer? =>

```
void f2() { }

void (*p)();  -> function pointer ka declaration
p = f2;
p();
```

e.g. =>

```
int (*p)(int);
p = f2;
p(5);

int f2(int x)
{

}.
```

#) Program =>

```
-> int f2(int);
main()
{
    int (*p)(int);
    p = f2;
    printf("%d", p(5));
    getch();
}
int f2(int x)
{
    printf("%d", x);
    return (x+2);
}
```

*