classmate
Date
Page

Programming Languages

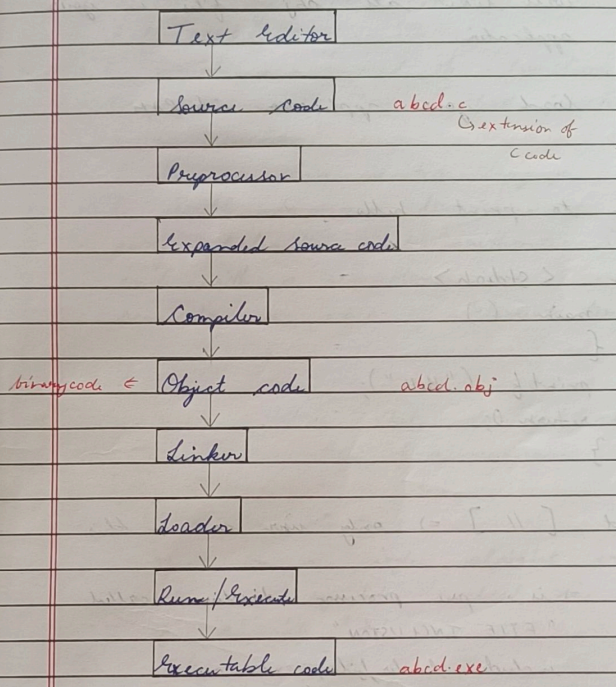
High Level
[closer to human/user]
- software
- application oriented
- Python, C++, Java

Low level
[closer to machine]
- hardware
- machine oriented
- Assembly, Binary (alphanumerics)

C is a middle level language
[both application and machine oriented]

header file => used to reduce complexity of codes & no. of lines. it is a file containing C declaration and definitions to be shared b/w several source files.

Processing of a Program
(steps are same in all languages)



Dev C++ => prefer for basics

Vs code => shows no errors

☆ **Preprocessor:** converts source code into expanded source code. It expands all the inbuilt functions that you are going to use in a program.

☆ **Compiler:** converts high level language to low level language or source code. It also checks for syntax errors.

☆ Syntax: way of writing commands

☆ Linker: will like all obj. files in your application

☆ Loader: load your application into RAM

// program to print hello

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

=) comment [//] => only user can see this

=) #include => is a pre processor directive called "FILE INCLUSION"
includes header files, extensions

=) stdio.h => standard input output header

input → scanf( )
output → printf( )

viva q → purpose of return 0

Introduction to C

- C is a middle level programming language developed by Dennis Ritchie in 1972 at AT&T Bell Labs, USA.
- C program is composed of functions and statements.
- C program must have atleast function i.e. called main function.
- C is a case sensitive language.
- C should be in lowercase.
- every declarative and arithmetic statement must end with a semicolon.
  semicolon is a.k.a. statement terminator in C.

☆ **Constants:** values that do not change during execution of a program.

type of constants in C:

① character constt.
must be a single alphabet or single digit or any symbol inclosed in a single quote
eg: 'A', '2', '+'.

② integer constt.
they are the whole numbers.
eg: 1, 50, 100, -2, ...

③ floating point constt.
decimal numbers. eg: 2.5, 3.7, ...
exponential " eg: $e^{3.2}$, ...

④ string constt.
collection of characters inclosed in double quotes. eg: "ABC", "123", "1+3"

☆ Variables
- values that may change during execution of a program.
- variables are the names given to a memory location.
- are also known as identifiers

how to declare a variable [variable declaration]
- variable declaration means allocating a memory to a variable.

datatype variable ;

int a; ⇒ 4 bytes
int a, b, c; ⇒ 12 bytes

data type [int, char, float]
- it is a type of info. that you're going to store in a memory.
- variables must be declared before use in a program.

integer words, int → 4 bytes
character words, char → 1 byte
floating point words, float → 4 bytes

variable initialization
initial value given to a variable.

☆ main function — entry point of program
main function is returning value to the operating system & it is returning 0 at the end of program and when 0 is recieved by the oper. system that means your program ran successfully till end of program.

O.S → made in C code, hence it is the operating system

if float then write return 0.0;

```
Ill: int main ()
{
    return 0;
}
```

return type

can be any int. like 1, 100, etc.

Shot on OnePlus

find area of circle use 3.14

online job

classmate
Date
Page

IM write a program for giving value to two numbers during compile time.

// sum of 2 nos (during compile time)

```
# include <stdio.h>
int main ( )
{
int a, b, c;
a = 5;
b = 10;
c = a + b;
printf ("%d", c);
return 0;
}
```

format string	
int	%d
char	%c
float	%f

→ gives output

Sum = 15

if you want output → 5 + 10 = 15

```
printf ("%d + %d = %d", a, b, c);
```

IM // Sum of 2 nos (during run time)

```
# include < >
int main ( )
{
int a, b, c;
scanf ("%d", &a);
scanf ("%d", &b);
c = a + b
printf ("Sum = %d", c);
printf ("%d + %d = %d", a, b, c);
return 0;
}
```

So that at output screen person gives input

address of

OR scanf("%d %d", &a, &b);

dont use &a in printf

III) // Perform all arithmetic operators (+, -, *, /)

```
#include <stdio.h>
int main ()
{
printf ("Enter 2 numbers:");
int a, b, c, d, e, f;
scanf ("%d %d", &a, &b);
printf ("Sum =
c = a+b;
printf ("Sum = %d", c);
printf ("Difference = %
d = a - b;
printf ("Difference = %d", d);
e = a*b;
printf ("Multiplication = %d", e);
f = a/b;
printf ("Division = %d", f);
return 0;
}
```

typecasting

if input is 5,2 then Div=2 if we want 2.5 as answer then float f; & "Division = %f"

[do c=a+b, d=... first & then use printf]

Output:

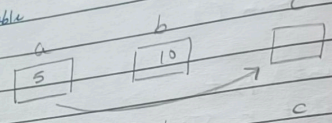
```
Enter 2 numbers:
_ _
Sum =
Difference=
Multiplication=
Division =
```

```
// simple interest
#include <stdio.h>
int main()
{
  int p, t;
  float r, SI;
  scanf("%d %lf %d", &p, &r, &t);
  SI = a*
  printf("Enter Principal: ");
  scanf("%d", &p);
  printf("Enter Rate: ");
  scanf("%f", &r);
  printf("Enter time: ");
  scanf("%d", &t);
  SI = (p * r * t) / 100;
  printf("SI = %f", SI);
  return 0;
}
```

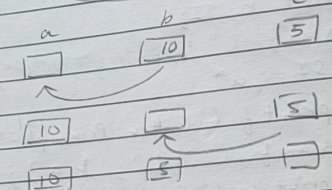
Output: Enter principal: 100

Classmate
Date
Page

Ill // write a program to swap two integers using 3rd variable



using 3rd variable:



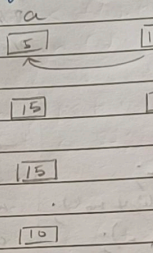

```
#include <stdio.h>
int main()
{
    int a, b, c;
    printf("Enter 2 nos");
    scanf("%d %d", &a, &b);
    c = a;
    a = b;
    b = c;
printf("After swapping a=%d, b=%d", a, b);
return 0;
}
```

right to left

method 2: // swap

#in

Method 2: // swap 2 integers without 3rd variable



a = a+b; a has value of both a+b

b = a-b;

a = a-b;

```
#include <stdio.h>
int main()
{
    int a, b;
    printf("Enter 2 nos");
    scanf("%d %d", &a, &b);
    a = a+b;
    b = a-b;
    a = a-b;
    printf("After swapping a=%d, b=%d", a, b);
    return 0;
}
```

classmate
Date
Page

% → mod operator / → quotient

% → remainder

// Program to find sum of all digits of a 3-digit no

I/P = (123) n

O/P = 1+2+3 = 6 dry run

n = 123

a = n%10;	a = 3
n = n/10;	→ 12
b = n%10;	12%10 → 2
n = n/10	12/10 → 1
c = n%10;	1%10 → 1

```
int main()
{
    int a,b,c,n,sum;
    printf("Enter 3 digit no");
    scanf("%d", &n);
    a = n%10;
    n = n/10;
    b = n%10;
    n = n/10;
    c = n%10;
    sum = a+b+c;
    printf("Sum of all digits = %d", sum);
    return 0;
}
```

if we need sum of squares

sum: a*a + b*b + c*c

% operator does not work with float values & works only with int values

// find reverse of a 3 digit number

I/p → 123
O/p → 321

same as previous but instead of sum
rev = a * 100 + b * 10 + c;
printf ("rev = %d", rev);
return 0;
}

// write a program to enter amount : 5869 and convert that amount into denominations from highest to lowest.

o/p: no. of 2000 notes → 2 (1869)
no. of 500 notes → 3 (369)
200 → 1 (169)
100 → 1

trial: int main ()
{
int amt, n2000, n500, n200, n100, n50, n20, n10, n5, n2, n1;
printf ("Enter amount:");
scanf ("%d", &amt);
n2000 = amt % 2000 ; → 2
n = n % 2000 ; 1869
n500 = amt % 500 ; → 3
n = n % 500 ; 369
n200 = amt % 200 ; → 1
n = n % 200 ; 169
n100 = amt % 100 ; → 1
n = n % 100 ; 69
n50 = amt % 50 ; → 1
n = n % 50 ; 19
n10 = amt % 10 ; → 1
n = n % 10 ; 9
n5 = amt % 5 ; → 1
n = n / 5 ; 4
n2 = amt % 2 ;

actual:

```
{
int amt, n2000, n500, n200, n100, n50, n20, n10, n5, n2, n1;
printf ("Enter amt:");
scanf ("%d", &amt);                                    → 2
n2000 = amt/2000;
amt = amt % 2000;                                      → 3
n500 = amt / 500;
amt = amt % 500;
n200 = amt/200;
amt = amt % 200;
n100 = amt/100;
amt = amt %100;
n50 = amt /50;
amt = amt%50;
n20 = amt%20;
amt = amt %10
n10 =
 .
 .
 .
n1 = amt/1;
amt = amt%1;
printf ("no. of 2000 note = %d", n2000);
                500
                200
                 .
                 .
                 .
}
```

LHS output

program name RMS

program

// write a program to enter basic salary : (b.s.)

HRA = 13% of basic salary

da = 14% of basic salary

travel allowance ta = 3% " " "

tax deduction td = 10% " " "

gross salary = bs + hra + da + ta − td => print

float

use (13*65)/100

☆ Types of Operator in C

operators : special symbols used for some mathematical & logical operations.

operands : those on which operators operate i.e. they can be consth. or variables.

① Arithmetic operators :

+ → Add

− → Subtract

* → Multiply

/ → quotient

% → mod (modulus) gives remainder

-5%-2 → -1

5%-2 → 1

-5%2 → -1

(5%-2 → 1)

5/2 → 2

-5/-2 → 2

5/-2 → -2

-5/2 → -2

5.0/2 → 2.5

5/2.0 → 2.5

50 % 2 → compiler error

NOTE :- % does not work with float variable.

difference b/w "=" & "==" => viva Q.

② Relational operators

= → assignment operator

>
>=
<
<=
!= (not equal to)
== (equality operators)

- relational operators work with if else condition and they are used for comparisons.
- always returns true or false

true → 1 false → 0

Eg: a=5, b=5

if(a==5) → 1 = true

a = 0

if(a=0) → false

if(a==0) → true

[here '=' is assignment.] => if (0) false.

but if(a=5) → true

③ logical operators

(for logical comparisons)

① && → logical and
② || → logical OR
③ ! → logical ~~not~~ NOT

They also return true or false

generally used in hardware devices

tve & -ve values ⟶ true value
0 ⟶ false value

&& ⇒ (multiplication)
returns true if both the conditions are true value

|| ⇒
if any of the conditions are true then it returns a true value

! ⇒ negate the expression
True ⟶ False
False ⟶ True

c = 5 1 1 0 (T T F)
c ⟶ True

c = 0 1 1 0
c ⟶ False

c = 5 && 2 (T T)
c ⟶ 1 [True]

-5 && 2 (T T)
c ⟶ 1 [True]

5 && 0 (T F)
c ⟶ 0 [False]

5 && 2 && 0
c ⟶ 0 [False]

a = 5 ⟶ T
while (!a)
≡] False

④ Bitwise operator
operates on expression or operand
works in bit
generally used in hardware devices

① Bitwise AND ⟶ &
② Bitwise OR ⟶ |
③ Bitwise XOR ⟶ ^
④ Bitwise 1's compliment ⟶ ~
⑤ Bitwise Left Shift ⟶ <<
⑥ Bitwise Right Shift ⟶ >>

a=5, b=12

first convert to bits

decimal to binary

2 | 5	
2 | 2 — 1	↑
1 — 0	

remainder

$(101)_2 = (5)_{10}$

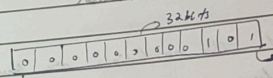
2 | 24	
2 | 12 → 0	
2 | 6 → 0	
2 | 3 → 0	
2 | 1 → 1	
0	

11000

$2^4 2^3 2^2 2^1 2^0$

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

$16 + 8 = 24$



reserved for sign (+/−)

0 1

bitwise AND a = 5 & 12

00000101

8 bit

00000100

8 4 2 1

4 + 8 = 12

00000101 ← 5

00001100 ← 12

& 00000100 multiplication

bitwise OR

00000101

| 00001100

00001101 addition

bitwise XOR (exclusive OR)

$\bar{A}B + A\bar{B}$

10 → T

01 → T, opp. true

11 → F

00 → F, same false

00000101

00001100

^ 00001001

classmate

Date ______
Page ______

bitwise 1's compliment

it converts bits from true (1) to false (0) & false (0) to true (1).

inverts the bits

a = 5 $(00000101)_2$

c = ~a — on only one operand

~5 = $(11111010)_2$

bitwise left shift (<<)

a << 2

↓ operand; └ no. of bits to be shifted

e. a = a << 2
printf("%d", a);

a = 5 $(00000101)_2$ << 2

shift ←

$(00010100)_2$ — 20

[Operand * $2^{\text{no. of bits}}$]

$5 \times 2^2 = 20$

bitwise right shift (>>)

a >> 3

a = 24 000 11000 >> 3

shift →

000 00011 — (3)

$\left[\frac{\text{operand}}{2^{\text{no. of bits}}}\right] = \frac{24}{2^3} = 8$

8 + 16

PYQs of minors
all semesters
part 3 - 4 years

classmate
Date
Page

Q.no

Typecasting (Type conversion)

converting one data type into another datatype.

Typecasting
- implicit
  automatically done by compiler
- explicit
  forcefully done by user using cast operator

Implicit

```
int a;
a = 5/2        → 2

float a;
a = 5/2        → 2.0

int a = 5.0/2
int a = 2.5 → 2

float a = 5.0/2   → 2.5
```

Explicit → using cast operator int changed to float

```
int a = float(5)/2
float ~~int~~ a = 5.0/2
float ~~int~~ a = 2.5   ~~2~~ 2
                        if int is used (implicit)
```

use examples in exams

data type chart google learn

classmate
Date
Page

Size of operator

it returns the no. of bits occupied by a datatype.

sizeof → single word

if you don't know size then

```
int a;
b = size of(a);
printf("%d", b);
```

output = 4

b = size of(char) → 1

size of('+') → 1

Conditional Statements

Syntax:

- if (condition) → no semicolon
  {
  do this
  }

- if (condition)
  {
  do this ← True
  }
  else
  {
  do this ← False
  }

// write a program to find greater of two numbers

```
#include main()
{
    int a, b;
    printf("Enter 2 numbers:");
    scanf("%d %d", &a, &b);
    if (a>b)  → condition
    { printf("a is greater");
    }
    else
    { printf("b is greater");
    }
    return 0;
}
```

→ If you want to print numbers
("%d is greater", a);

// write a program to check whether number is even or odd

```
{
int a;
printf ("Enter a no:");
scanf ("%d",&a);
if (a%2 == 0)
 {
  printf ("%d is even", a);
 }
else
 {
  printf ("%d is odd", a);
 }
return 0;
}
```

// check whether number is positive or negative

```
{
int a;
printf ("Enter a no:");
scanf ("%d", &a);
if (a > 0)
 {
  print f ("%d is positive", a);
 }
else
 {
 printf ("%d is negative", a);
 }
return 0;
}
```

Char ⇒ single alphabet only

// write a program to enter any character check whether that character is a vowel or not

```
int main ()
{
char ch;
printf ("Enter character:");
scanf ("%c", &ch);
if ( (ch == 'a') || (ch == 'e') || (ch == 'i') || (ch == 'o') ||
     (ch == 'u'))
{ printf ("vowel");
}
else
{
printf ("not a vowel");
}
return 0;
}
```

A ≠ 'A'
variable const.

HW check whether year is leap year or not
divisible by 4 but not by 100
exactly 4 in it.

logical and
logical or
400, 100, 4

```
if ((y%4==0) && (y%100!=0) || (y%400==0))
{
printf ("leap year");
}
```

classmate
Date
Page

multiple else-if (else-if ladder)

```
if (condition 1)
 {
  do this
 }
elseif (condition 2)
 {
  do this
 }
else if (cond^n 3)
 {
 }
else
 {do this
 }
```

true then it will not check other cond^n's

else if space

// find greatest among 3 numbers

```
{int a,b,c;
 printf (" Enter 3nos : ");
 scanf ("%d %d %d", a,b,c);
 if ( (a>b) && (a>c))
  {
   printf ("A is greatest");
 else if ((b>a) && (b>c))
  {
   printf ("B greatest");
  }
 else
  {
   printf ("C greatest");
  }
```

// write a program to enter 3 sides of a triangle and check whether it is equilateral, isoceles or scalene

```
int main()
{ int a, b, c;
  printf("Enter 3 sides:");
  scanf("%d %d %d", a, b, c);
  if ((a==b) && (b==c))
    {
      printf("This is equilateral");
    }
  else if(((a==b) && (b!=c)) || ((b==c) && (c!=a)) ||
          ((c==a) && (a!=b)))
    {
      printf("Isoceles Triangle");
    }
  else if ((a!=b) && (b!=c) && (c!=a))
    {
      printf("Scalene Triangle");
    }
  return 0;
}
```

HW a program to enter marks of 5 subjects, find its total & avg. if avg >= 80 then print grade A

>= 60-80 grade B

40-60 grade C

<40 fail

```
int main ()
{
    float m1, m2, m3, total, avg;
    scanf ("%f %f %f", &m1, &m2, &m3);
    total = m1+m2+m3;
    avg = total / 3;

    if (avg >= 80)
    {
    printf ("Grade A");
    }
    else if ((avg >= 60) && (avg < 80))
    {
    printf ("B");
    }
    else if ((avg >= 40) && (avg < 60))
    {
    printf ("C");
    }
    else
    {
    printf ("Fail");
    }
    return 0;
}
```

III) write a program to enter any character, check whether that character is uppercase, lowercase, a symbol, or a digit.

ASCII codes

classmate
Date
Page

American standard code for international interchange
[ASCII codes]

- character cannot be directly converted to binary & decimal can be converted.
- as each character has a decimal value.

A - Z ⟶ 65-90
a - z ⟶ 97-122
0 - 9 ⟶ 48-57

```
int main ()
{
    char ch;
    scanf ("%c", &ch);
    if ((ch >= 65) && (ch <= 90))
    {
    printf ("upper case");
    }
    else if ((ch >= 97) && (ch <= 122))
    {
    printf ("lower case");
    }
    else if (ch >= 48) && (ch <= 57))
    {
    printf ("digit");
    }
    else
    {
    printf ("Any other symbol");
    }
    return 0;
}
```

if we want to check ASCII of a char

```
scanf ("%c", &ch);      → input : D
printf ("%c", ch);      → output : D
printf ("%d", ch);      → output : 68
```

// enter any characters, check lowercase / uppercase
if its uppercase, convert it into lowercase & vice versa

```
int main()
{
  char ch;
  scanf ("%c", &ch);
  if ((ch>=65)&&(ch<=90))
  {
    printf ("uppercase");
    ch = ch + 32;
    printf ("lowercase= %c", ch);
  }
  else if ((ch >=97) && (ch <=122))
  {
    printf ("lowercase");
    ch = ch - 32;
    printf ("uppercase = %c", ch);
  }
  return 0;
}
```

A, B ... Z
65 66 ... 90
+32 (diff)
a, b ... z
97 98 ... 122
-32

NOTE: if we dont know ASCII codes

```
((ch >= 'A') && (ch <= 'Z'))
& ((ch >= 'a') && (ch <= 'z'))
```

or ('0' & '9')

HW: Δ problem with nesting

classmate
Date
Page

#

NESTED if else

if within if

// greatest of 3 digits using nesting

```
int main ()
{
    int a,b,c;
    scanf("%d%d%d", &a, &b, &c);
    if (a>b)
    {
        if (a>c)
        {
        printf("A");
        }
        else
        {
        printf("C");
        }
    }
    else if (b>c)
        {
        printf("B");
        }
    else
        {
        printf("C");
        }
    return 0;
}
```

classmate
Date
Page

Looping Statements

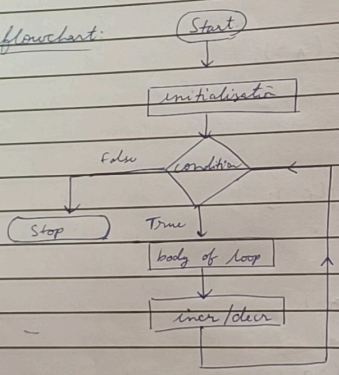
Types: ① while loop
② for loop
③ do - while loop

While Loop

Syntax:

```
initialisation;
while (condition)
{
    body of loop
    incr/decr;        increment/decrement
}
```

flowchart:



increment operator (++)

it is a unary operator [operates on single operand]
↳ eg: &, ~, !
(address)
binary operator eg: +, -, /, &&, ||

adds '1' to the value of operand

(++)

post increment
a++;
a=a+1;

pre increment
++a;
a=a+1;] here both ++a & a++ are same

but c=(a++)+b a=5, b=2

[c = a+b ; 5+2=7
a = a+1 ; 6]

a=6, b=2, c=7

c=(++a)+b;

[a = a+1 ; 6
c = a+b ; c=6+2=8]

a=6, b=2, c=8

Ill c=(a++)+(a++)+(b++)+b; [all post op]

c = a + a+1 + b + b

a = a+1

a = a+1

b = b+1

[Solⁿ:
c = 14
a = 7
b = 3]

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

post pre post pre

first pre a = 5 a+1 6

b = 2 b+1 3

a is overwritten ← c = 6 a + 6 a + 3 b + 3 b = 18

then post a = 6 a+1 7

b = 3 b+1 4

[a=7, b=4, c=18]

varies per compiler
try on vscode

decrement operator (--) a=a-1;
unary operator

post (a--;)
pre (--a;)

// write a program to print hello 10 times.

```
int main()
{
    int i;
    i = 1;                    => initialisation
    while(i<=10)
    {
        printf("Hello\n");    -> to print in new line every time
        i++;
    }
    return 0;
}
```

i=10, i>=1, i--

// print numbers 1 to 10

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

// print even numbers from (1 to 100)

```
int main();
{
  int i;
  i = 2;
  while (i <= 100)
  {
   printf ("%d\n", i);
  i = i+2;
  }
  return 0;
}
```

// write a program to print table of a user entered number

```
  int main()
  {
    int i, n, t;
    scanf ("%d", &n);
    i = 1;
    while (i <= 10)
    {
      t = n * i;
    printf ("%d * %d = %d\n", n, i, t);
     i = ++;
    }
    return 0;
  }
```

n	i	t
5	x 1	= 5
5	x 2	= 10
5	x 3	= 15
.	.	.
5	x 10	= 50

classmate

Date
Page

27/4

// write a program to find sum of 10 natural numbers

1 + 2 + 3 + ... + 10 =

i = 1, ... 10

i	Sum i Sum
i = 1	0 + 1 = 1
i = 2	1 + 2 = 3
i = 3	3 + 3 = 6
i = 4	6 + 4 = 10

```
{
    int i, s = 0;
initialisation ← i = 1;
    while (i <= 10)
    {
        s = s + i;
        i++;
    }
    printf (" 1+2+... +10 = %d", s);
    return 0;
}
```

HW: 2 + 4 + ... + 100 = ?

// factorial of a number

5! = 5 × 4 × 3 × 2 × 1

f * i = f

i = 5 f = 1 × 5 = 5

i = 4 f = 5 × 4 = 20

i = 3 f = 20 × 3 = 60

i = 2 f = 60 × 2 = 120

i = 1 f = 120 × 1 = 120

```
{
int i, n, f = 1;
scanf ("%d", &n);
i = n;
while (i >= 1)
{
    f = f * i;
    i--;
}
printf ("factorial = %d", f);
}
```

// write a program to find power a^b using while loop.

```
int main ()
{
    int a, b, p=1, i;
    printf ("Enter base & power");
    scanf ("%d %d", &a, &b);
    i = 1;
    while (i <= b)
    {p = p * a;
     i++;
    }
    printf ("power = %d", p);
} return 0;
```

// find reverse of n digit number

```
int main ()
{
    int n, rem, rev=0, s=0;
    printf ("Enter n-digit no:");
    scanf ("%d", &n);
    while (n!=0)
    {
        rem = n%10;
        rev = rev*10 + rem;
        s = s+rem;
        n = n/10;
    }
    printf ("Reverse = %d", rev);
    printf ("Sum = %d", s);
    return 0;
}
```

=> in the previous we did this step till n/10 = 0

like the previous program

		rev*10 + rem
n = 1234	rem = 4	0*10 + 4 =
n = 123	rem = 3	4*10 + 3 =
n = 12	rem = 2	43*10 + 2 =
n = 1	rem = 1	432*10 + 1 =

sum of squares of all digits => s = s + (rem * rem)

// check whether number is palindrome or not.

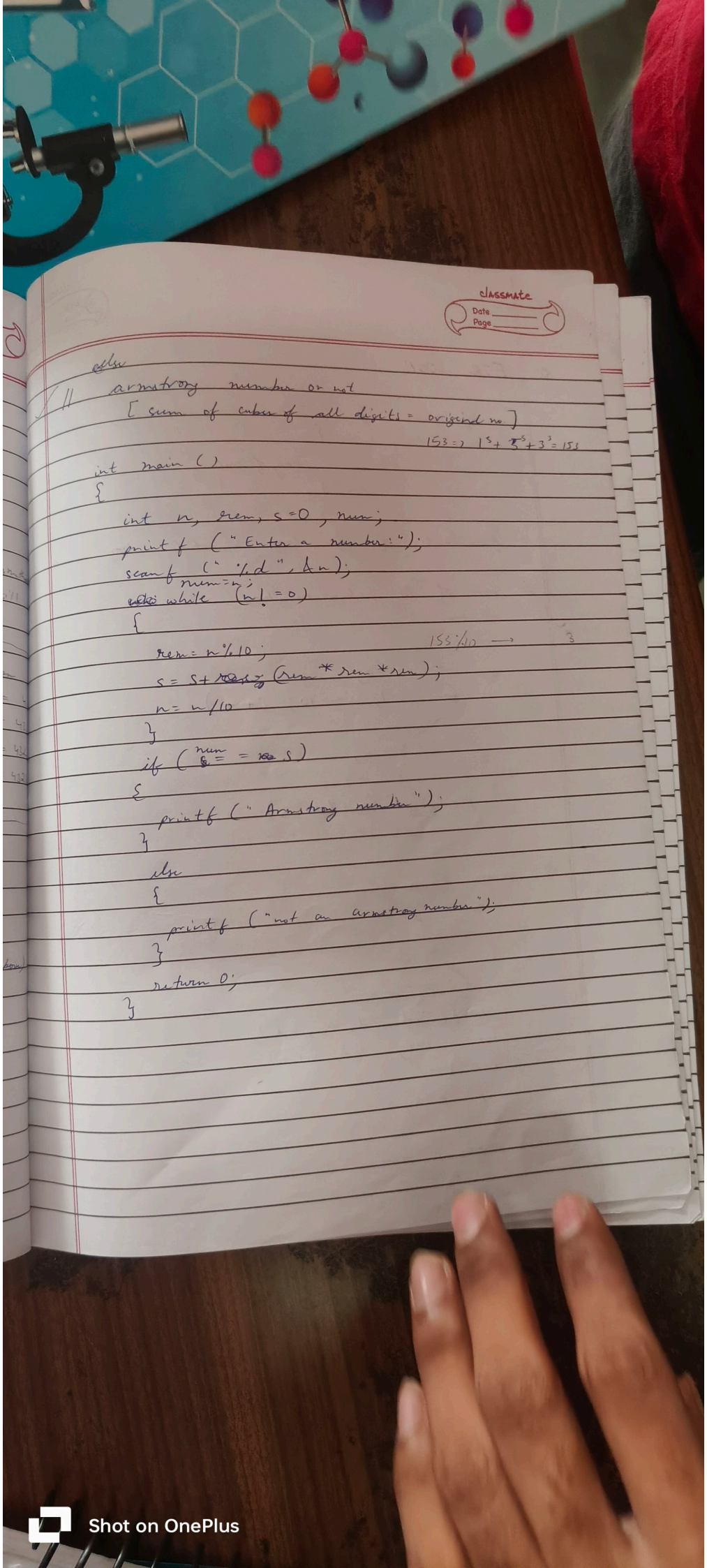
first find rev like above (till line 12 of above) introduce a new int => num.

[int n, rev=0, rem, num] in line 3 above

& n in line 6, num = n

then after while loop

```
if (num == rev)
{
    printf ("palindrome");
}
```



```
else
// armstrong number or not
   [ sum of cubes of all digits = original no ]
                                  153 => 1^3 + 5^3 + 3^3 = 153

int main ()
{
   int n, rem, s=0, num;
   printf ("Enter a number:");
   scanf ("%d", &n);
   num=n;
   while (n!=0)
   {
      rem = n%10;                          153%10 ⟶ 3
      s = s+ (rem * rem * rem);
      n = n/10
   }
   if (num == s)
   {
      printf ("Armstrong number");
   }
   else
   {
      printf ("not an armstrong number");
   }
   return 0;
}
```

P.Y. Question → draw flowchart of every loop

☆ FOR LOOP

syntax : for (initialisation; condition; incr/decrement)
{
body of code
}

everything in same line

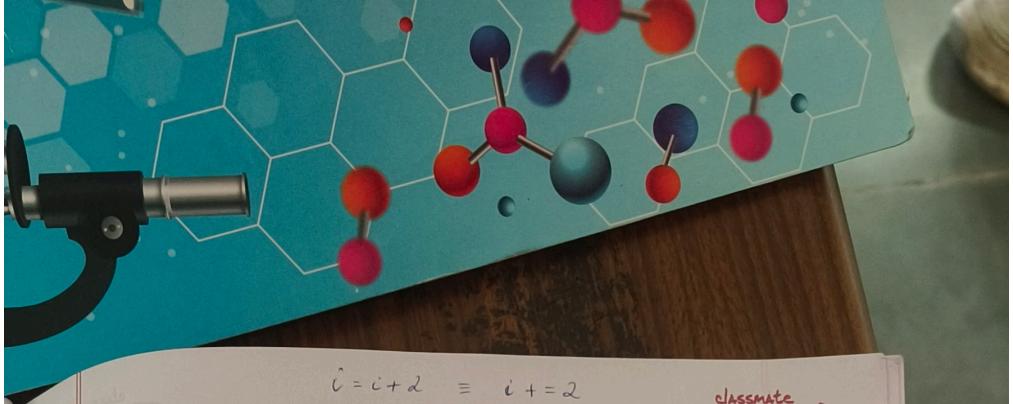
flowchart of for loop & while loop is similar

// sum of 10 natural numbers

```
int main ()
{
    int i, s=0;
    for (i=1; i<=10; i++)
    {
        s = s + i;
    }
    printf ("sum = %d", s)
}
```

// power of no.

```
int main ()
{
    int i, p=1, a, b;
    scanf("%d %d", &a, &b);
    for (i=1; i<=b; i++)
    {
        p = p*a;
    }
    printf ("power = %d", p);
```



$i = i + 2 \equiv i += 2$

classmate
Date
Page

// factorial

```
int main()
{
    int i, f=1, n;
    scanf
    for ( i=n; i>=1; i--)
    {
        f = f * i;
    }
    printf ("factorial = %d", f);
}
```

// check whether number is prime number or not

```
main()
{
int n, i;
scanf ("%d", &n);
for (i=2; i<=n-1; i++)
{
    if (n%i==0)
    {
    printf ("not prime")
    break;
    }
}
if (i==n-1)
{
printf ("prime")
}
}
```

looping is a.k.a. iterative statements

10 loops => 10 iterations

Nested for [for within for]

↳ use for printing rows & columns

```
for ( initialisation; condition; incr/decr)
{
    for ( initialisation; condition; incr/decr)
    {
        body of inner loop
    }
    body of outer loop
}
```

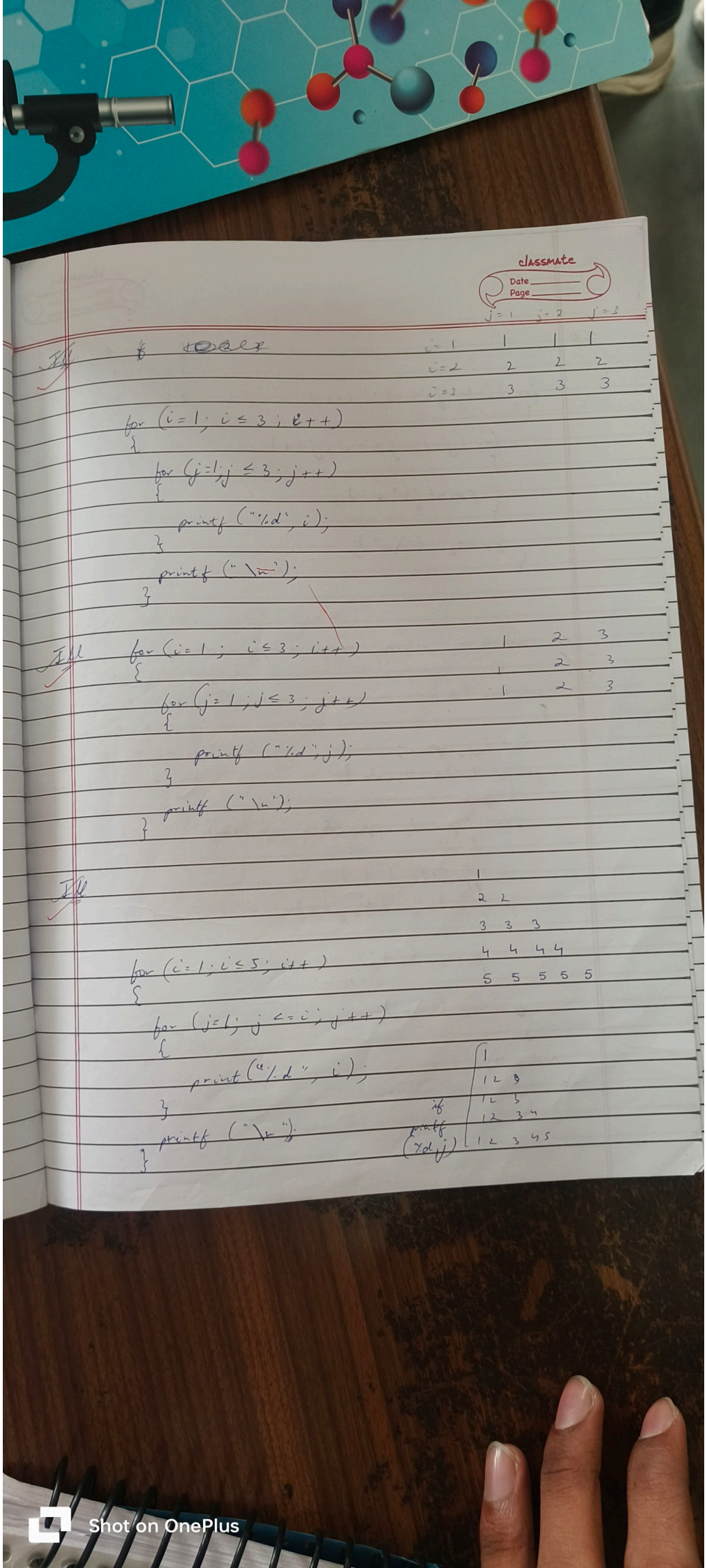
III
```
for (i=1; i<=3; i++)
{
    for (j=1; j<=3; j++)        ] loops 9 times
    {
        =
    }
}
```

I/II print rows & columns of stars

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

output

* * * *
* * * *
* * * *
* * * *


Classmate
Date
Page
j=1 j=2 j=3
i=1 1 1 1
i=2 2 2 2
i=3 3 3 3
for (i=1; i<=3; i++)
{
for (j=1; j<=3; j++)
{
printf ("%d", i);
}
printf ("\n");
}
1 2 3
1 2 3
1 2 3
for (i=1; i<=3; i++)
{
for (j=1; j<=3; j++)
{
printf ("%d", j);
}
printf ("\n");
}
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
for (i=1; i<=5; i++)
{
for (j=1; j<=i; j++)
{
printf ("%d", i);
}
printf ("\n");
}
if printf (%d, j)
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```
for (i=1; i<=5; i++)
{
    for (j=1; j<=i; j++)
    {
        printf("*");
    }
    printf("\n");
}
```

// write a program to print pyramid of stars.

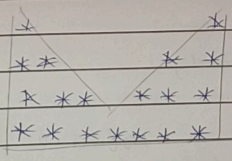
```
int main()
{
    int i,j,space;
    for (i=1; i<=5; i++)
    {
        for (space=1; space<=5-i; space++)
        {
            printf(" ");
        }
        for (j=1; j<=i; j++)
        {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

n = 5	space
i = 1	4
i = 2	3
i = 3	2
i = 4	1
i = 5	0

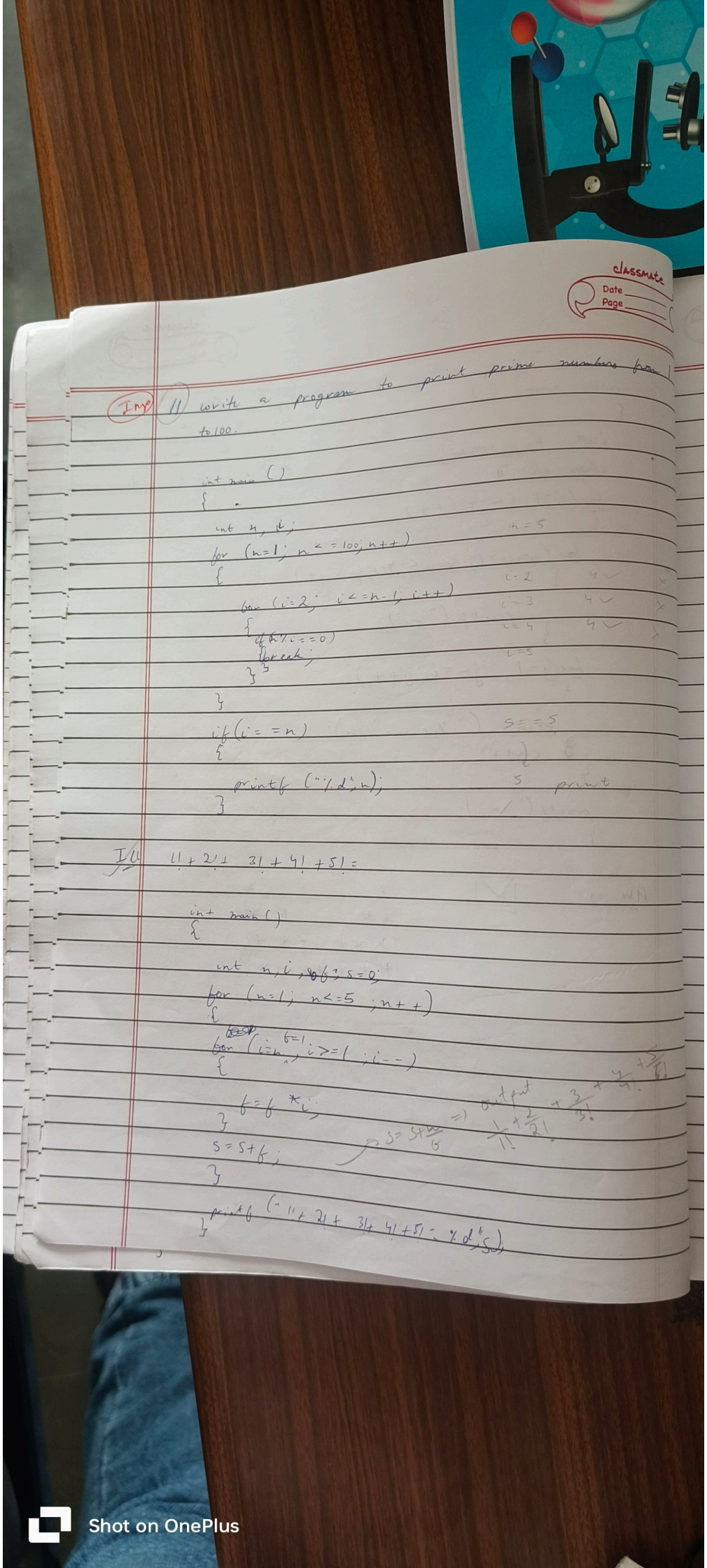
```
int main ()
{
int i, j, space;
char ch = 'A';
for(i=1; i<=4; i++)
{
  for (space=1; space<=4-i; space++)
  {
    printf (" - ");
  }
for(j=1; j<=i; j++)
{
  printf ("%c " ch);
  ch++
}
printf ("\n");
}
```

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

HW =>



```
* _ _ _ _ _ *
* * _ _ _ * *
* * * _ * * *
* * * * * * *
```

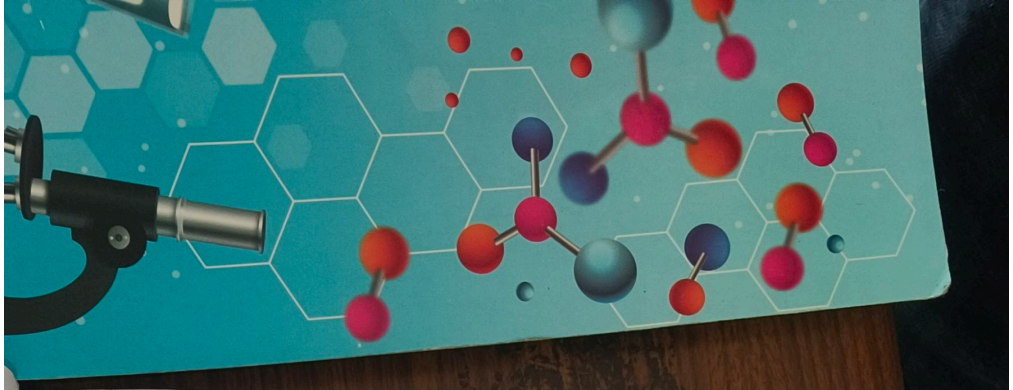


IMP // write a program to print prime numbers from 1 to 100.

```
int main ()
{
  int n, i;
  for (n=1; n<=100; n++)
  {
    for (i=2; i<=n-1; i++)
    {
      if (n%i==0)
        break;
    }
    if (i==n)
    {
      printf ("%d", n);
    }
  }
```

I/O 1! + 2! + 3! + 4! + 5! =

```
int main ()
{
  int n, i, f, s=0;
  for (n=1; n<=5; n++)
  {
    f=1
    for (i=n; i>=1; i--)
    {
      f=f*i;
    }
    s=s+f;
  }
  printf ("1! + 2! + 3! + 4! + 5! = %d", s);
}
```



classmate
Date
Page

HW Q sol:

brea

☆ Jumping statements

① break

it is a keyword used to break the running loop

```
for (i=1; i<=10; i++)
{
printf ("Inside the loop");
if (i==4)
{break}
printf ("%d", i);
}
```

Inside the loop
1
Inside the loop
2
Inside the loop
3
Inside the loop

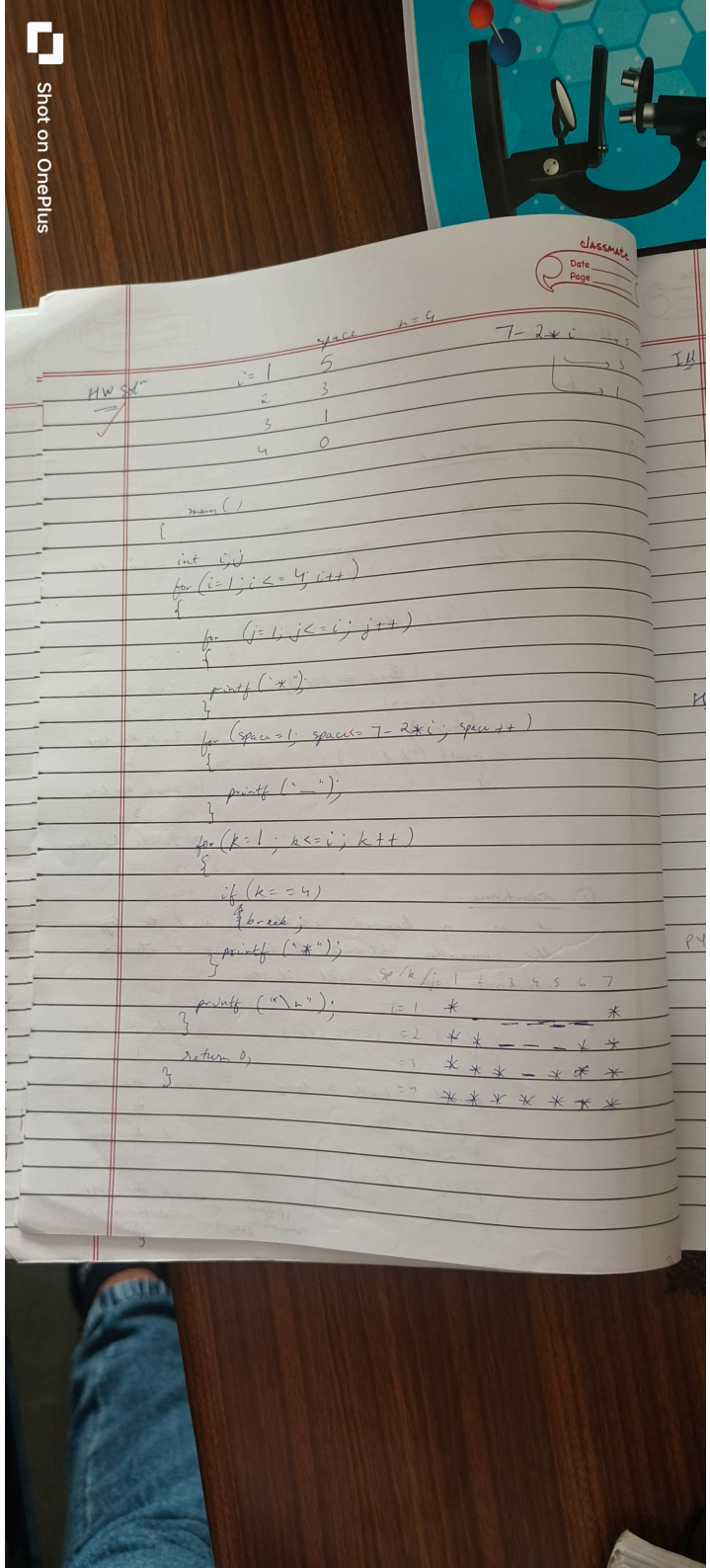
② continue

it is a keyword which is used to bypass the remainder statements of the current iteration & continue with the next iteration

```
for (i=1; i<=10; i++)
{
printf ("Inside the loop
if (i==4)
{ continue;}
printf ("%d", i);
}
```

inside the loop
1
inside the loop
2
inside the loop
3
inside the loop
inside the loop
5

skipped remainder statement


Shot on OnePlus
classmate
Date
Page
HW Soln
n = 4
space
i = 1 5
2 3
3 1
4 0
7 - 2 * i
main()
{
int i,j;
for(i=1;i<=4;i++)
{
for (j=1;j<=i;j++)
{
printf("*");
}
for (space=1; space<=7-2*i; space++)
{
printf(" ");
}
for(k=1; k<=i; k++)
{
if(k==4)
{break;}
printf("*");
}
printf("\n");
}
return 0;
}

Ill $\frac{1^1}{1} + \frac{2^2}{2!} + \frac{3^3}{3!} + \frac{4^4}{4!} + \frac{5^5}{5!}$

∴ $1! + 2! + \cdots$ prog.

add another for loop:

```
for(j=1; j<=n; j++)
{
    p=p*n;
}
s=s+p/f;
}
```

```
s=0, p=1
for(i=1; i<=n; i++)
{ p=p*n;
  if (p%2!=0)
     s=s+p;
=> PYQ
  else s=s-p;
}
```

HW $n - n^2 + n^3 + -n^4 + \cdots \; n^n = 1$

$n = ?$

$n = ?$

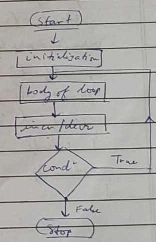
Do-while loop

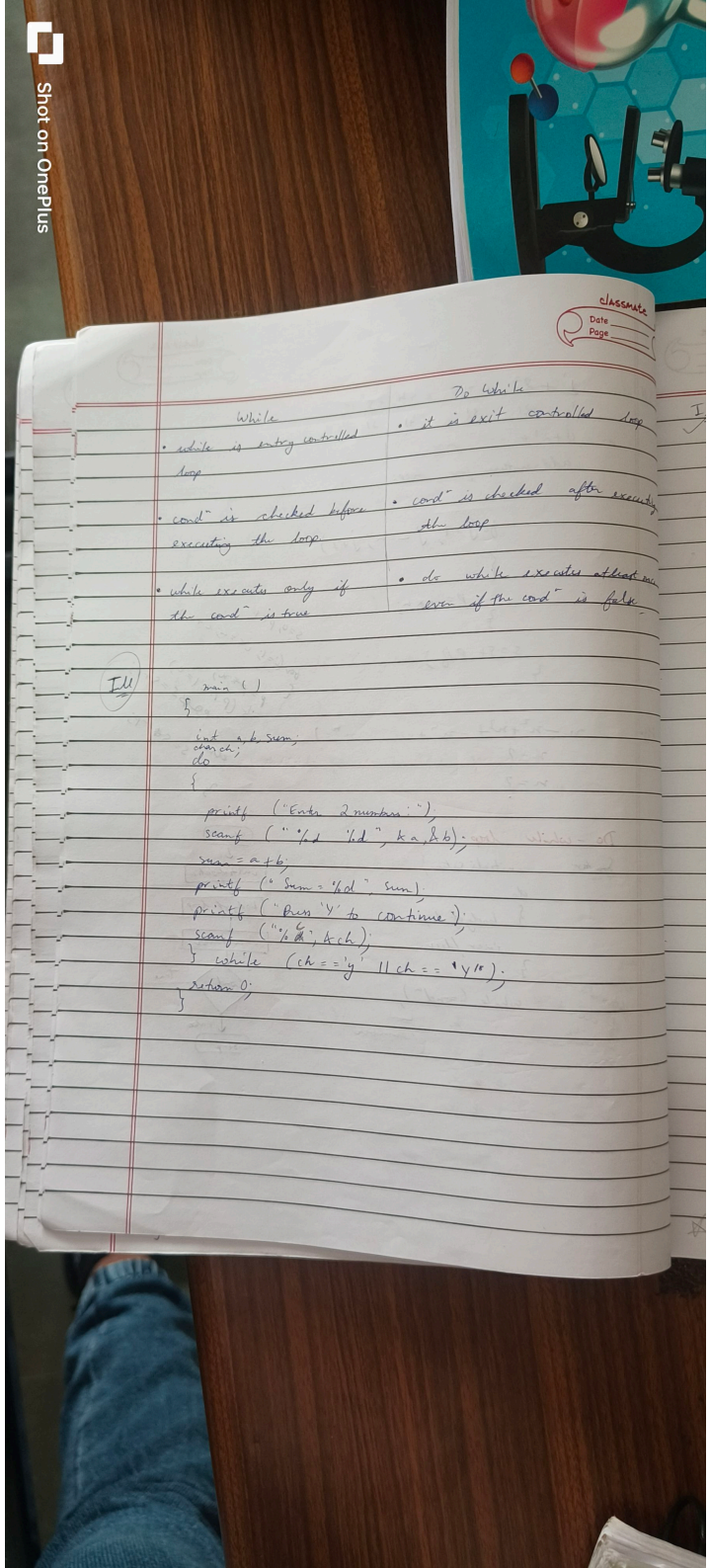
Syntax:

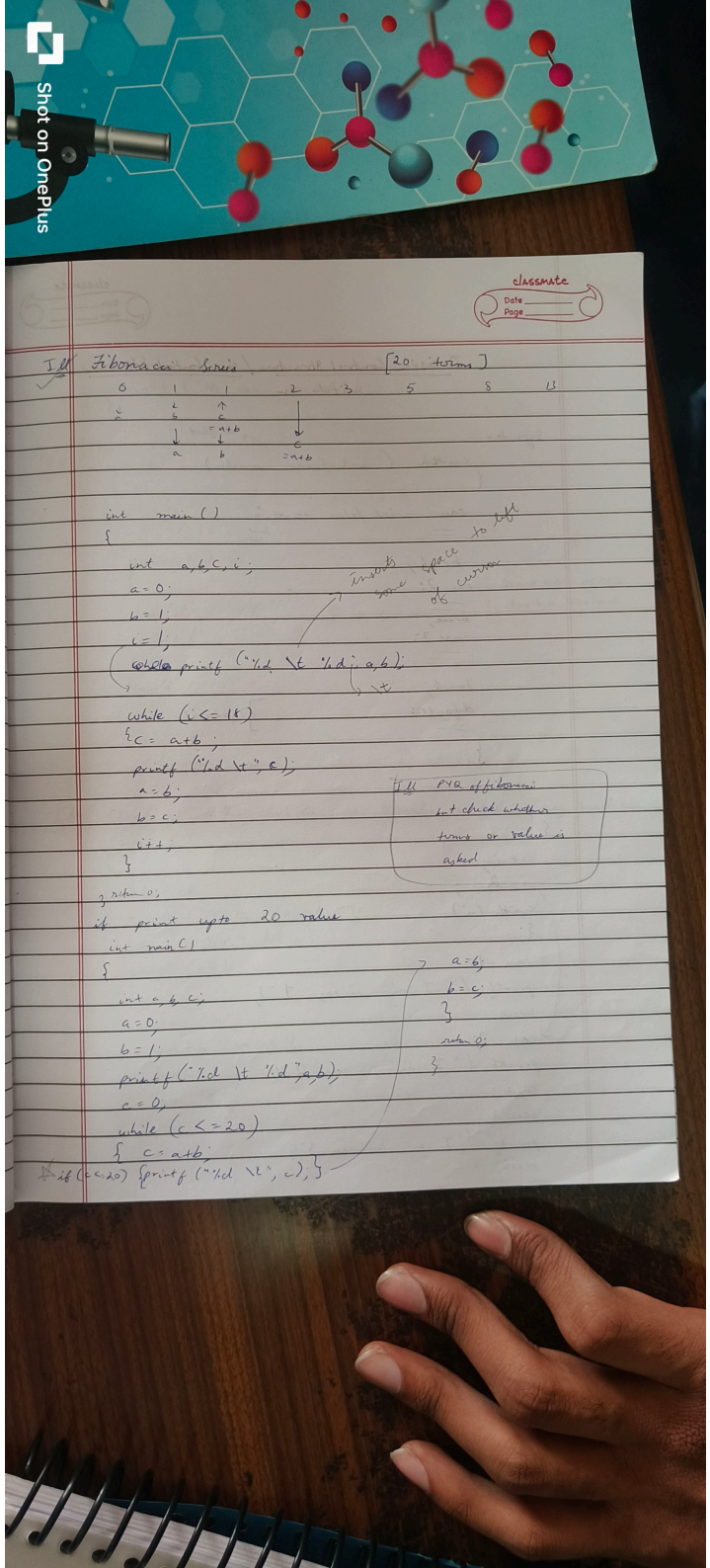
```
initialisation;
do
{ body of loop
  incr/decr
}
while (condn);
```

remember

PYQ => diff b/w flowcharts of do-while & while




Shot on OnePlus
classmate
Date
Page
While
• while is entry controlled loop
• cond" is checked before executing the loop
• while executes only if the cond" is true
Do While
• it is exit controlled loop
• cond" is checked after executing the loop
• do while executes atleast once even if the cond" is false
III
main ()
{
int a, b, sum;
char ch;
do
{
printf ("Enter 2 numbers:");
scanf ("%d %d", &a, &b);
sum = a+b;
printf ("Sum = %d", sum);
printf ("Press 'y' to continue:");
scanf ("%c", &ch);
} while (ch == 'y' || ch == 'Y');
return 0;
}


classmate
Date
Page
Fibonacci Series [20 terms]
0 1 1 2 3 5 8 13
int main()
{
int a,b,c,i;
a=0;
b=1;
i=1;
printf("%d \t %d",a,b);
while (i<=18)
{c=a+b;
printf("%d \t",c);
a=b;
b=c;
i++;
}
return 0;
if print upto 20 value
int main()
{
int a,b,c;
a=0;
b=1;
printf("%d \t %d",a,b);
c=0;
while (c<=20)
{ c=a+b;
if (c<=20) {printf("%d \t",c);}
a=b;
b=c;
}
return 0;
}
PYQ of fibonacci but check with no terms or value is asked

classmate
Date
Page

Decision Control Structure / case Control Structure

switch case

Syntax:

```
switch ( int / char variable )        // no float
{
case int / char const. :              // colon
   ==
   break;
case 2:
   ==
   break;
case 3:
   ==
   break;
default:                              // optional
   ==
}
```

every case must end with a break

```
main ()
{
int i;
i = 2 ;
switch (i)
{
case 1:
printf ("I am in case 1");
break;
case 2:
printf ("I am in case 2");
break;
case 3:
printf ("I am in case 3");
break;
```

```
default :
printf ("No case match");
}
```

II

```
main ()
{
 char i;
 i = '2';
 switch (i)
 {
  case '1':
  printf (`          `);
 break;
 case '2':
   :
   :
  like above program.
```

III Check whether character is vowel or not using switch case

```
main ()
{
 char ch;
 printf ("Enter character:");
 scanf ("%c", &ch);
 switch (ch)
 {
   case 'a';
   printf ("vowel");
   break;
   case 'e';
   printf ("vowel");
   break;
```

```
case 'i':
printf ("vowel");
break;
case 'o':
printf ("vowel");
break;
case 'u':
printf ("vowel");
break;
default:
printf ("Not a vowel");
}
return 0;
}
```

or =)

```
case 'a': case 'e': case 'i': case 'o': case 'u': case 'A'
case 'E': case 'I': case 'O': case 'U':
printf ("vowel");
break;
default:
printf ("Not a vowel");
}
return 0;
}
```

Q) write a menu driven program to make calculator using switch case.

+ - * /

```
#include <stdio.h>  ] write both
#include <stdlib.h>

int main()
{
do
{
int a, b, sum, diff, i;
char ch;
printf("Enter 2 nos:");
scanf("%d %d", &a, &b);
do
{
printf("Press 1 for addition");
printf("Press 2 for sub");
printf("Enter your choice");
scanf("%d", &i);
switch(i)
{
case 1:
sum = a+b;
printf("Sum = %d", sum);
break;
case 2:
diff = a-b;
printf("diff = %d", diff);
break;
case 3:
exit(0);
default:
printf("Enter valid choice");
}
} while(i!=3);
return 0;
}
```

[printf("Press 3 to enter nos");] ← 3 → case if we want to start with new nos.

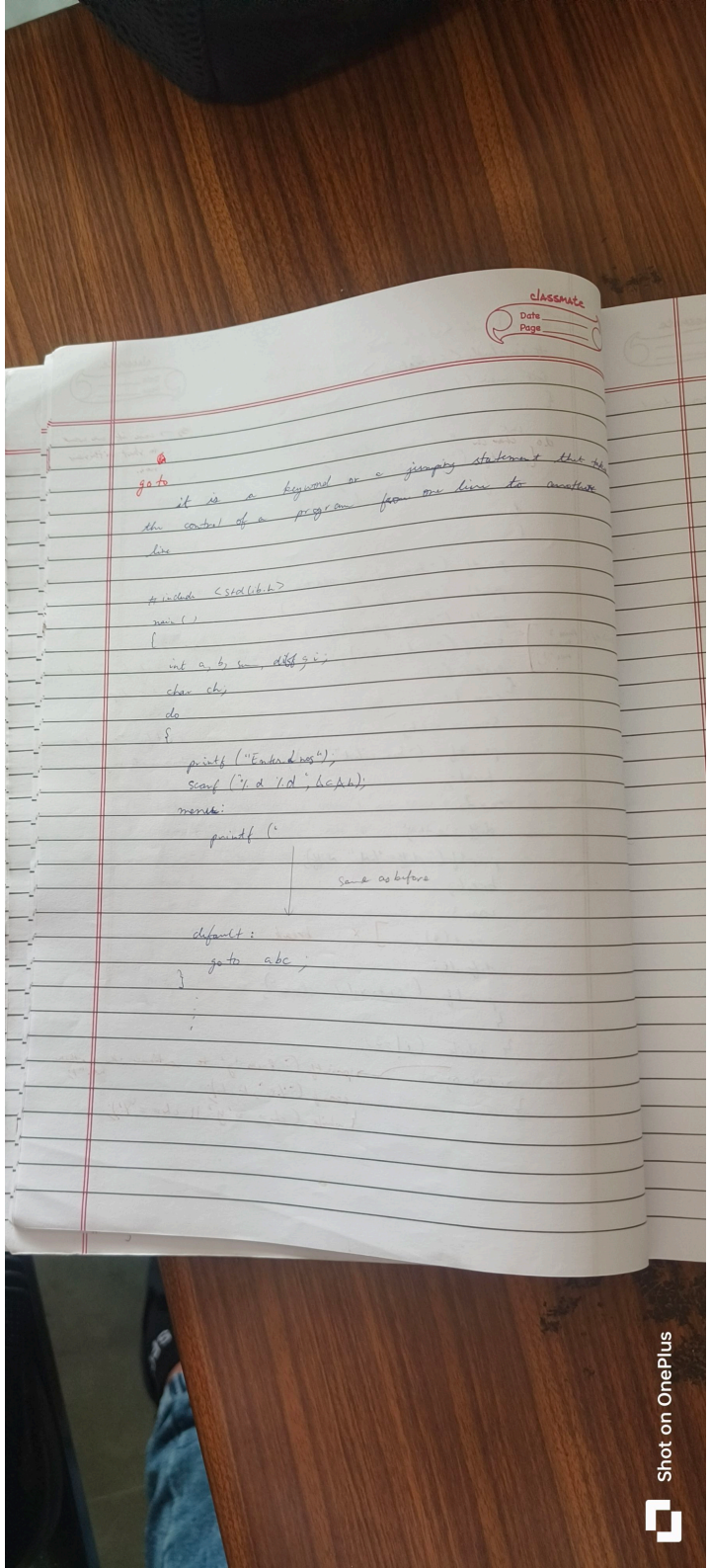
→ 4 div
→ 5 start addition

break func used to exit a loop

exit func used to terminate a program

] x break

```
printf("Press 'y' to continue with another no.");
scanf("%c", &ch);
} while(ch=='y' || ch=='Y');
```


classmate
Date
Page
goto
it is a keyword or a jumping statement that transfers the control of a program from one line to another line
#include <stdlib.h>
main()
{
int a, b, sum, diff, i;
char ch;
do
{
printf("Enter two nos");
scanf("%d %d", &a, &b);
menu:
printf("
same as before
default:
goto abc;
}
;
Shot on OnePlus

Rules for declaring variable names

- every var name must start with an alphabet not with digit
- space & . is not allowed.
- _ is allowed any where, even in start

Key words

are the pre defined words whose meaning has already been defined to compiler.
They are the reserved words so they can't be used as variable names.
There are 32 keywords in C

Tokens

it is the smallest unit of program. It can be a const, variable, operator.

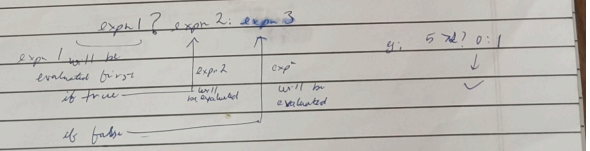
int a; a = 1
↓ new ↓↓↓ tokens

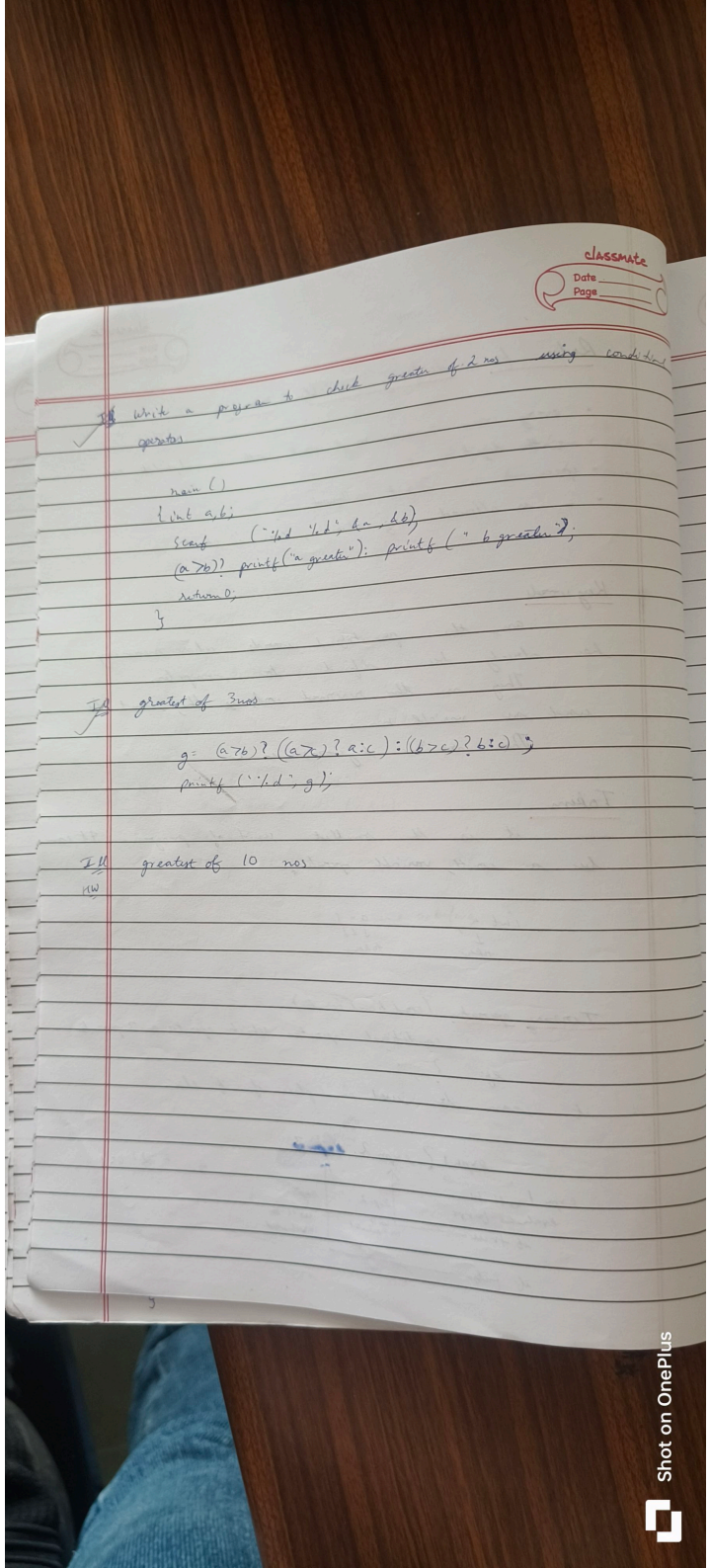
Ternary operator (conditional operator)

conditional operator which operates on 3 operands

eg: ? , :

it can be used in place of if else.




classmate
Date
Page
1) Write a pgm to check greater of 2 nos using cond. op
main ()
{ int a,b;
scanf ("%d %d", &a, &b);
(a>b)? printf("a greater"): printf ("b greater");
return 0;
}
2) greatest of 3 nos
g= (a>b)? ((a>c)? a:c):((b>c)? b:c);
printf ("%d", g);
3) greatest of 10 nos
HW
Shot on OnePlus

Array

a homogeneous collection of elements of same data type

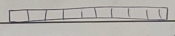
→ array always start from zero
→ the starting address of an array is called base address
→ array elements are always stored in adjacent memory location

Declaration of an array

int a[10]; → a is an array variable which can store 10 int.

memory { 1 int → 4 bytes
10 int → 40 bytes

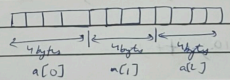
we need a base address to get location.

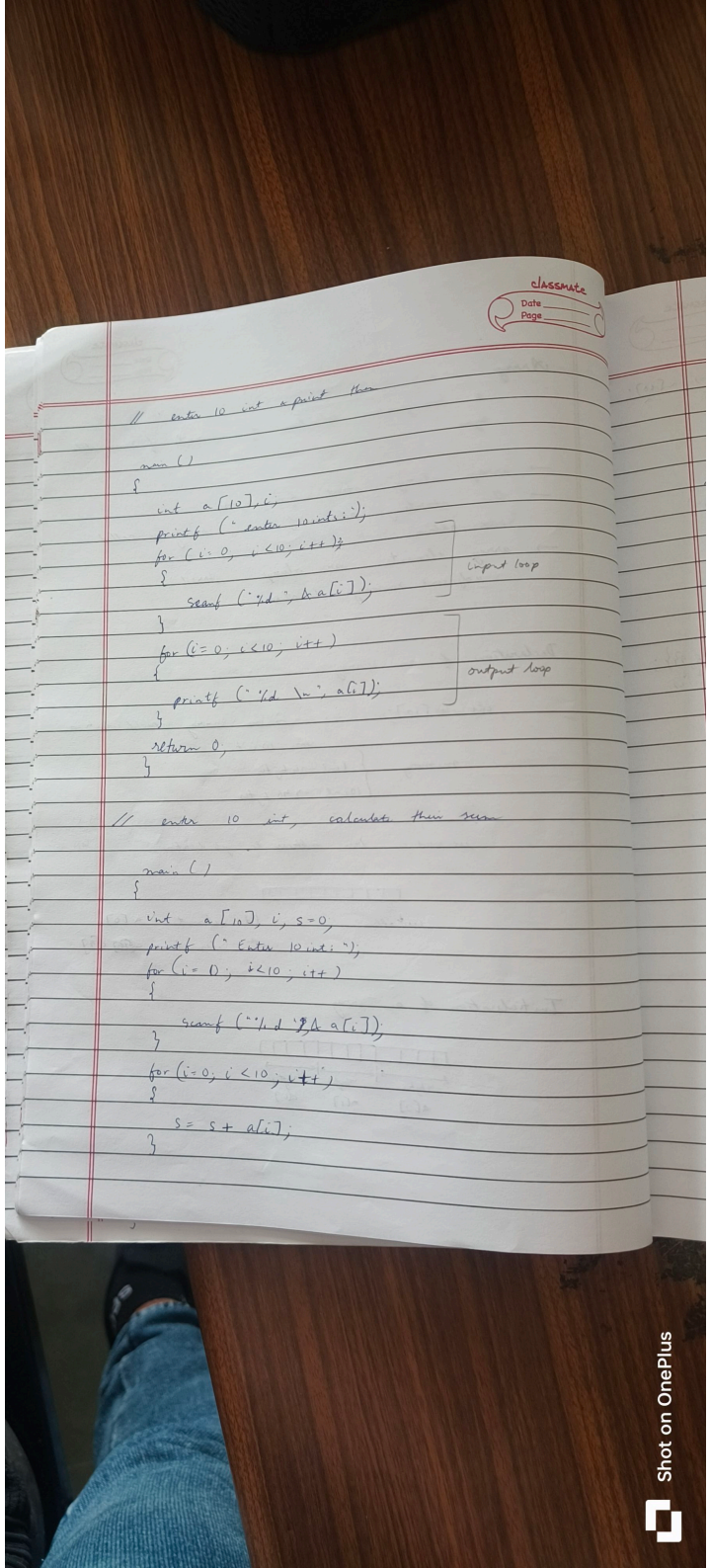


base address

1st element → a[0]
10th element → a[9]

Initialisation of an array





// enter 10 int & print them

```
main()
{
  int a[10], i;
  printf("enter 10 ints:");
  for(i=0; i<10; i++);
  {
    scanf("%d", &a[i]);
  }
  for(i=0; i<10; i++)
  {
    printf("%d \n", a[i]);
  }
  return 0;
}
```

input loop

output loop

// enter 10 int, calculate their sum

```
main()
{
  int a[10], i, s=0;
  printf("Enter 10 int: ");
  for(i=0; i<10; i++)
  {
    scanf("%d", &a[i]);
  }
  for(i=0; i<10; i++)
  {
    s = s + a[i];
  }
```

```
    printf ("sum of all ints : %d", s);
    return 0;
}

// Sum of even & odd int

main()
{ int a[10], i, se = 0, so = 0;
  printf ("Enter 10 int:");
  for (i=0; i<10; i++);
  {
     scanf ("%d", &a[i]);
  }
  for (i=0; i<10; i++)
  {
        if (a[i] % 2 == 0)
        { se = se + a[i]}
        else
        { so = so + a[i];
        }
  }
  printf ("even int sum = %d", se);
  printf ("odd int sum = %d", so);
  return 0;
}
```

to find sum at even positions write (i % 2 == 0)

\t => 8 spaces
[tab]

classmate
Date
Page

// write a program to enter a 3x3 matrix & print matrix in a matrix form

```
main()
{
    int a[3][3], i, j;
    printf("Enter matrix");
    for (i=0; i<3; i++)
    {
        for (j=0; j<3; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    for (i=0; i<3; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d \t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

input

output

output

a00	a01	a02
a10	a11	a12
a20	a21	a22

Shot on OnePlus

// find sum of all elements of a matrix

```
main()
{
int a[3],[3], i,j, s=0;
printf("Enter matrix");
```

put matrix

```
for(i=0; i<3; i++)
{
  for(j=0; j<3; j++)
  {
    s = s + a[i][j];
  }
}
printf("%d", s);
return 0;
}
```

s_i s_j

// find sum of diagonal elements

add =) if (i==j)
{ s = s + a[i][j]

if (i>j) => upper △
(i<j) => lower △

// find sum of diagonal elements 2 matrices

```
main()
{
  int a[3][3], b[3][3], i, j, c[3][3];
  printf("Enter 1st matrix");
```

input 1

```
  printf("Enter 2nd matrix");
```

input 2

```
  for(i=0; i<3; i++)
  {
    for(j=0; j<3; j++)
    {
      c[i][j] = a[i][j] + b[i][j];
    }
  }
```

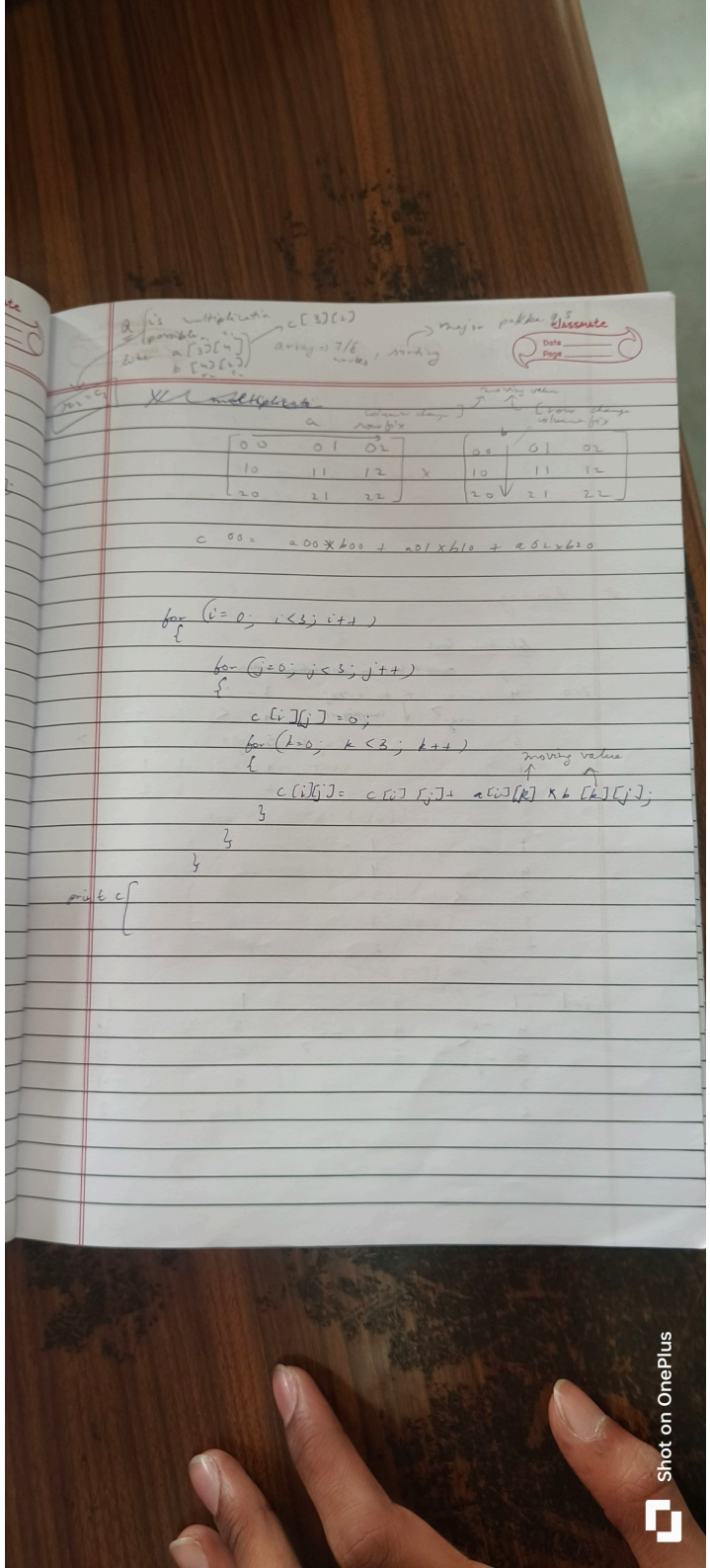
code for sum

print c matrix

// multiplication of 2 matrices

input 1

input 2



classmate
Date
Page

II PYQ → find determinant

$$D = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

use formula itself

$$D = \begin{bmatrix} a[ab] \times (a_{11} \times a_{22} - a_{12} \times \\ \\ \end{bmatrix}$$

☆ Sorting

Selection Sort

array — sort in ascending order

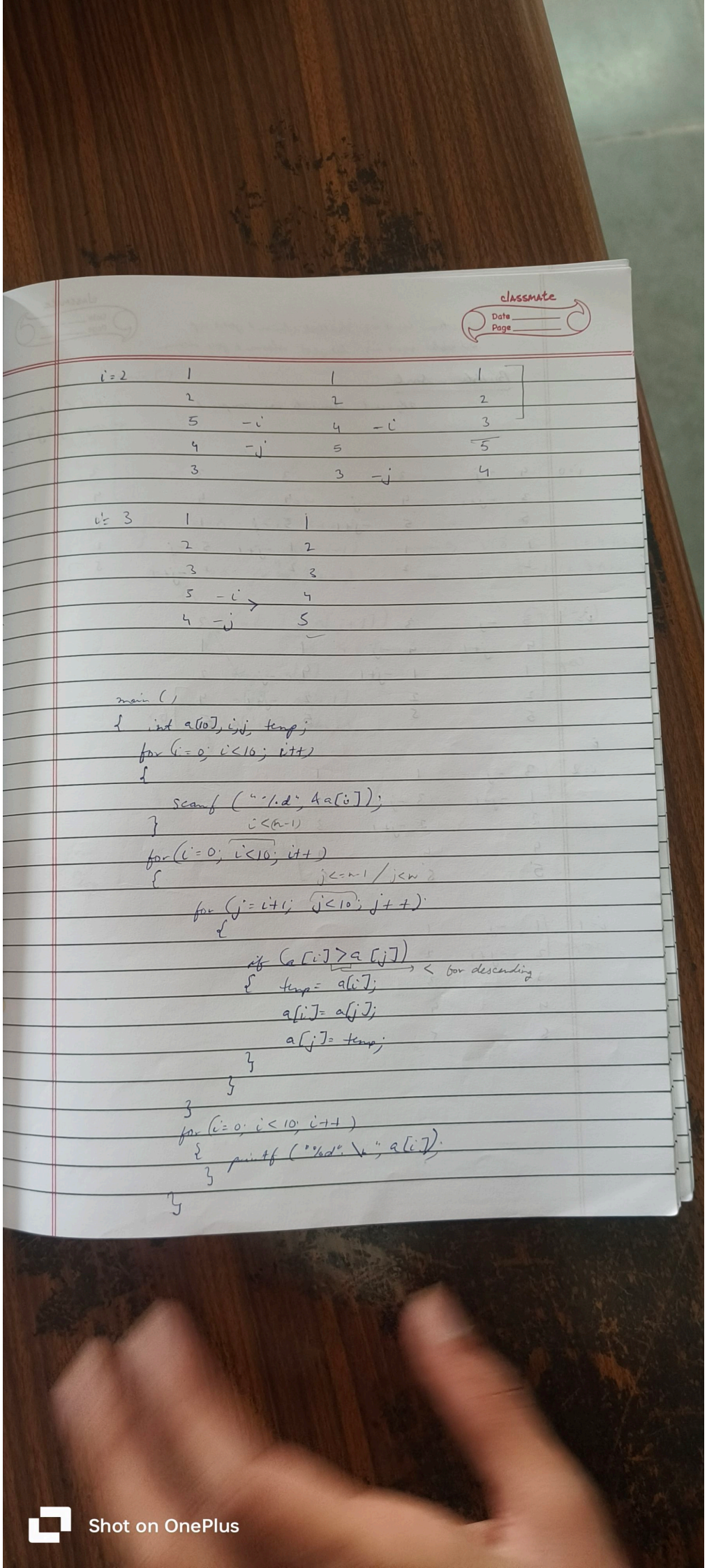
i=0

4	→ a_i	>	as $a_i > a_j$ swapping will	3 – i	3 – i
3	→ a_j			4 <	4
5				5 – j	5 >
1				1	1 – j
2				2	2

1 – i		1	← fixed now
4		4	
5	<	5	
3		3	
2 – j		2	

i=1

1		1		1		1	
4 – i	<	4	– i	3	– i	2	
5 – j		5	>	5		5	
3		3		4		4	
2		2	– j	2	– j	3	



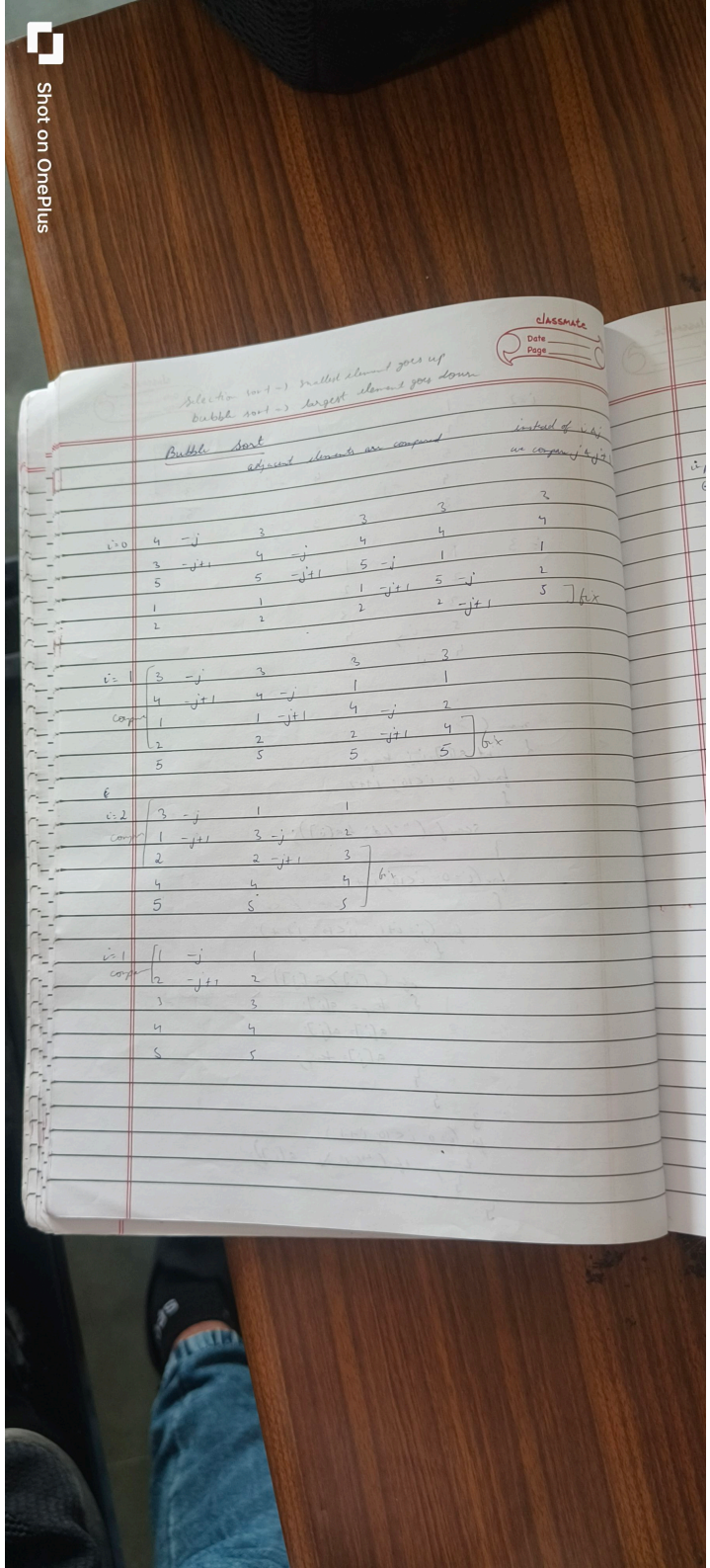
i=2

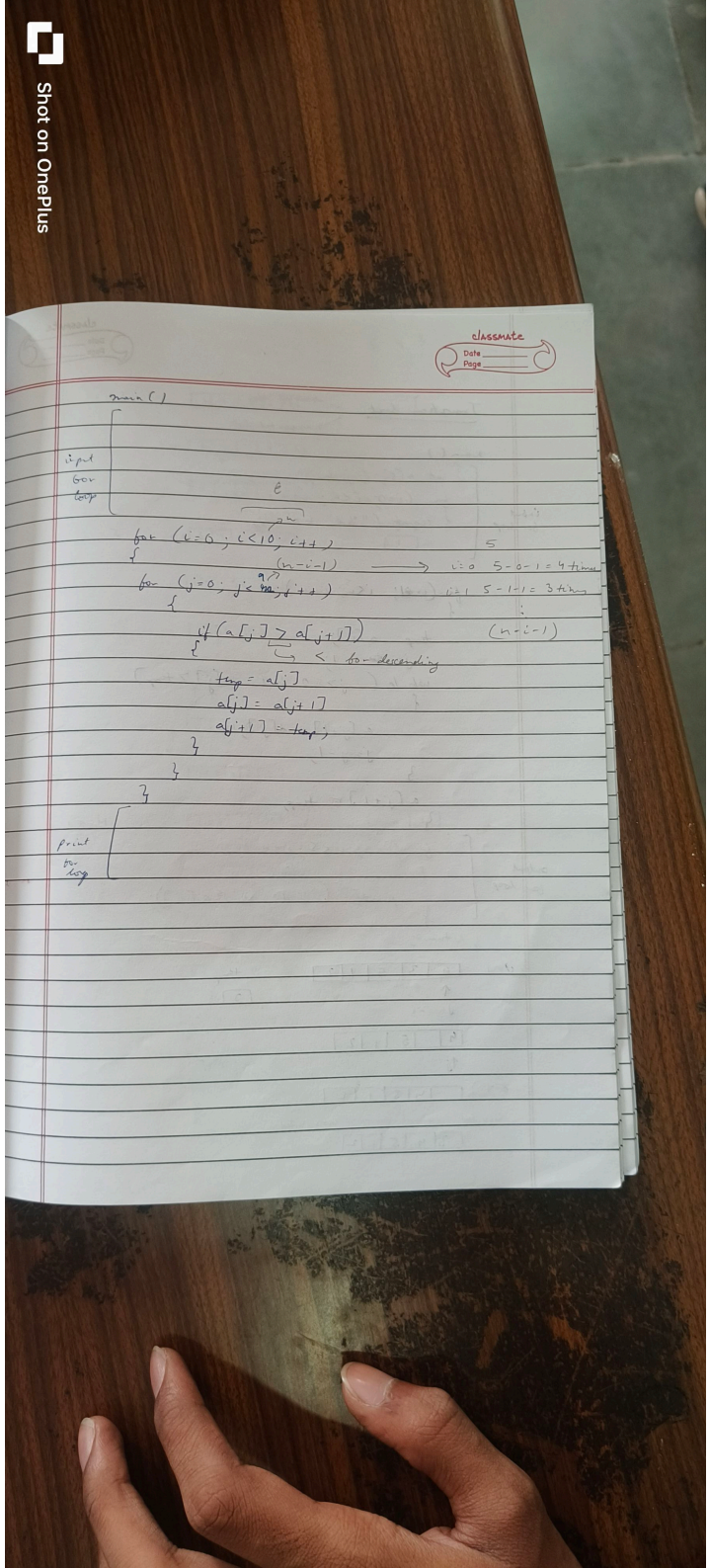
1		1		1
2		2		2
5	-i	4	-i	3
4	-j	5		5
3		3	-j	4

i= 3

1		1	
2		2	
3		3	
5	-i →	4	
4	-j	5	

```
main()
{  int a[10], i,j, temp;
   for(i=0; i<10; i++)
   {
      scanf("%d", &a[i]);
   }
                 i<(n-1)
   for(i=0; i<10; i++)
   {
                          j<=n-1 / j<n
      for(j=i+1; j<10; j++)
      {
         if(a[i]>a[j])        < for descending
         {  temp= a[i];
            a[i]= a[j];
            a[j]= temp;
         }
      }
   }
   for(i=0; i<10; i++)
   {
      printf("%d", a[i]);
   }
}
```


Shot on OnePlus
classmate
Date
Page
Selection sort → smallest element goes up
bubble sort → largest element goes down
Bubble Sort
adjacent elements are compared
instead of i, j
we compare j & j+1
i=0
i=1
compare
i=2
compare
i=1
compare
fix
fix
fix


Shot on OnePlus
classmate
Date
Page
main ()
input
for
temp
for (i=0; i<10; i++)
(n-i-1)
for (j=0; j<n; j++)
if (a[j] > a[j+1])
< to descending
temp = a[j]
a[j] = a[j+1]
a[j+1] = temp;
print
for
loop
5
i:0 5-0-1 = 4 times
i:1 5-1-1 = 3 times
(n-i-1)

Insertion Sort → sorted list, → unsorted list

```
main()
{ int a[5], i, j, temp;
  for (i=0; i<5; i++)
  { scanf("%d", &a[i]); }

for(i=1; i<5; i++)
{
  temp = a[i];
  j = i-1;
  while ( j>=0 && a[j] > temp)
  {
    a[j+1] = a[j];
    j = j-1;
  }
  a[j+1] = temp;
}
for (i=0; i<5; i++)
{
  printf("%d \n", a[i]);
}
return 0;
}
```

i=1 for loop

output for loop

i=1 | 4 | 3 | 5 | 1 | 2 | temp = 3

j = i-1

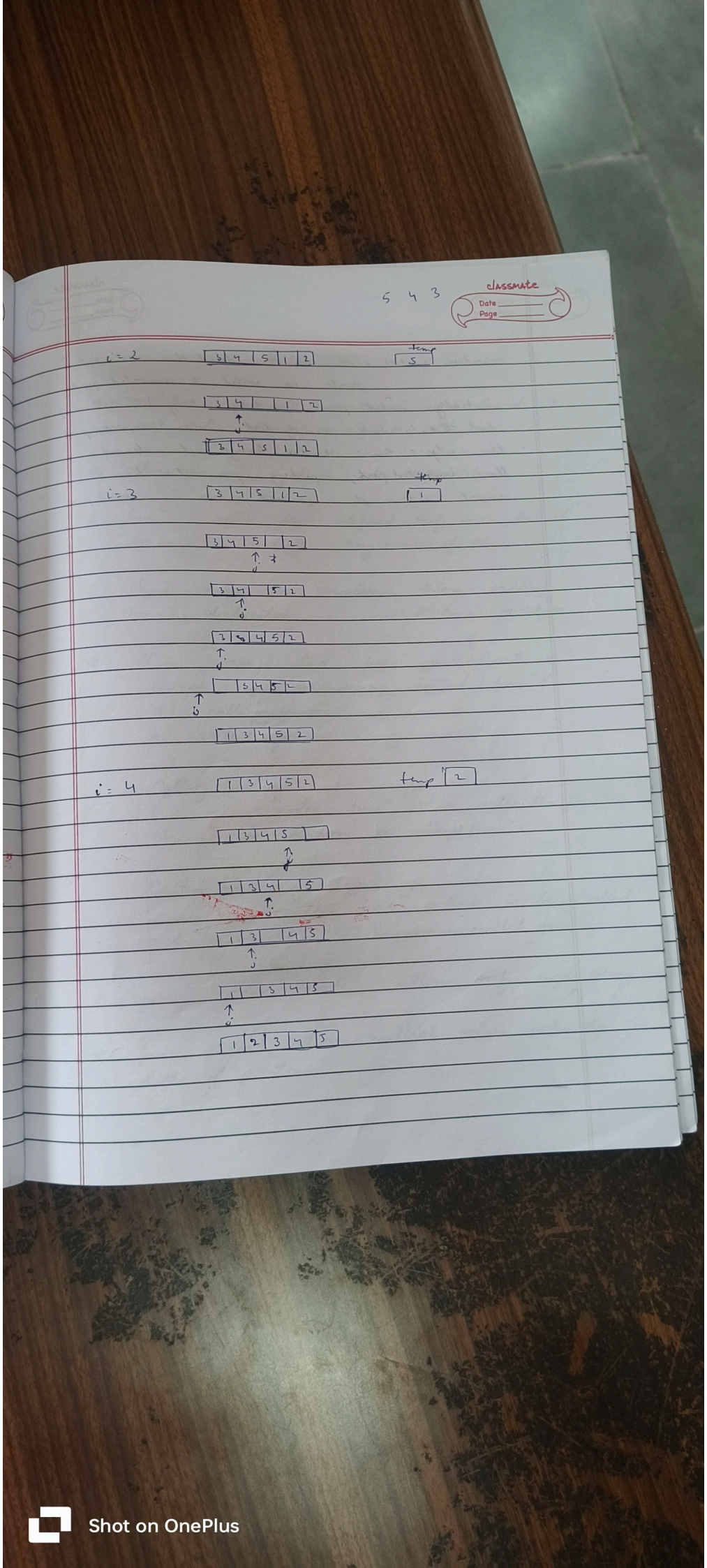
| 4 | 5 | 1 | 2 |

↑j

j=-1 | 4 | 3 | 1 | 2 |

sorted ← → unsorted

| 3 | 4 | 5 | 1 | 2 |


5 4 3
classmate
Date
Page
i = 2
temp
i = 3
temp
i = 4
temp

insertion sort =) • idea is to divide the list into 2 parts : as sorted & unsorted part.

• initially sorted part contains only 1st element of list while rest of list is in unsorted part.

• the algo then iterates through each element in the unsorted part, picking up one at a time & insert into its correct position in sorted part

FUNCTIONS

* Function is a self contained block of code to carry out some specific task.

* Advantages of func's are reusability.

* func will be defined only once, but it can be called any number of times.

* Func must be declared before calling. Func declaration is called func prototype.

* There are two func's to execute a concept of a user defined func's:
  a.) calling func
  b.) called func

Calling func:
The func which calls the another func is known as calling func.

Called func:
The func which is being called by another func is called called func

* func can return only single value.

* func can return multiple values with the help of pointers.

classmate
Date ____
Page ____

★ By default, func^n returns integer value

★ The arguments or parameters in a calling func^n are called actual func^n's arguments.

★ the arguments or parameters in a called func^n are known as formal parameters.

Func^n declaration / Func^n prototype — learn syntax

return-type (space) fn name (list of arg / parameters);
separated by ,

① no return type no arg
② no return type with arg
③ with return type with arg
④ with return type no arg

I No return type no arg

no operation
no input needed
no output given

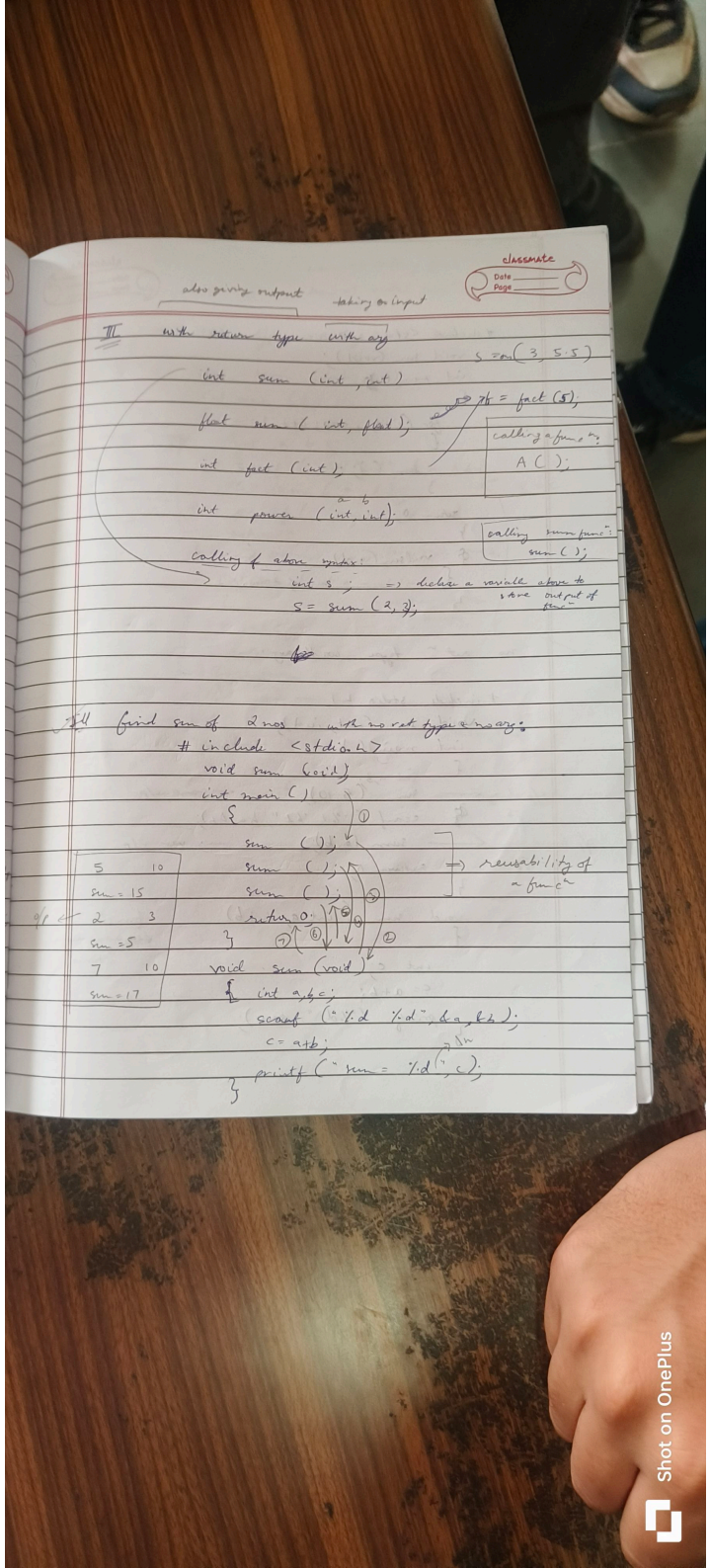
void A (void); → can also be written as void A ( );

II No return type with arg

void sum (int, int); → no. of inputs taken
void sum (int, float, int);

void fact (int); → we are taking only one input in factorial syntax.

calling
sum (a,b) ← var.
→ sum (2, 3), ← const
→ sum (2, 3.5, 5) o/p


classmate
Date
Page
also giving output
taking an input
III. with return type with arg
int sum (int, int)
float sum (int, float);
int fact (int);
int power (int, int);
a b
calling of above syntax:
int s; => declare a variable above to store output of fun^n
s = sum (2, 3);
s = sum (3, 5.5)
yf = fact (5);
calling a fun^n in fun^n:
A ( );
calling sum fun^n:
sum ( );
5 10
sum = 15
2 3
sum = 5
7 10
sum = 17
=> reusability of a fun^n
#include <stdio.h>
void sum (void)
int main ( )
{
sum ( );
sum ( );
sum ( );
return 0;
}
void sum (void)
{ int a,b,c;
scanf ("%d %d", &a, &b);
c = a+b;
printf ("sum = %d", c);
}

```
#include <stdio.h>
void sum (void);
int main ()
{
    for (i=1; i<=10; i++)
    {
        sum ();
    }
    return 0;
}
void sum (void)
{ ---- sum of 2 nos.
}
```

func^{n} runs 10 times

no ret^{n} type with arg:

```
#include stdio.h)
void sum (int, int);
int main ()
{   int a,b;
    sum (5,10);
    scanf ("%d %d", &a, &b);
    sum (a,b);
    return 0;
}
void sum (int a, int b)
{
    int c;
    c = a+b;
    printf ("%d", c);
}
```

o/p always is — const i/p

o/p based on i/p — var. i/p

func^{n} runs 2 times

main => calling func^n

sum => called func^n

with ret. type with arg :

```
int sum (int, int);
int main ()
{
    int a, b, s;
    scanf ("%d %d", &a, &b);
    s = sum (a, b);
    printf ("sum = %d", s);
    return 0;
}
```

actual arguments (a, b)

s = c

ends program

for storing a and b,

```
int sum (int a, int b)
{
    int c;
    c = a+b;
    return c;
}
```

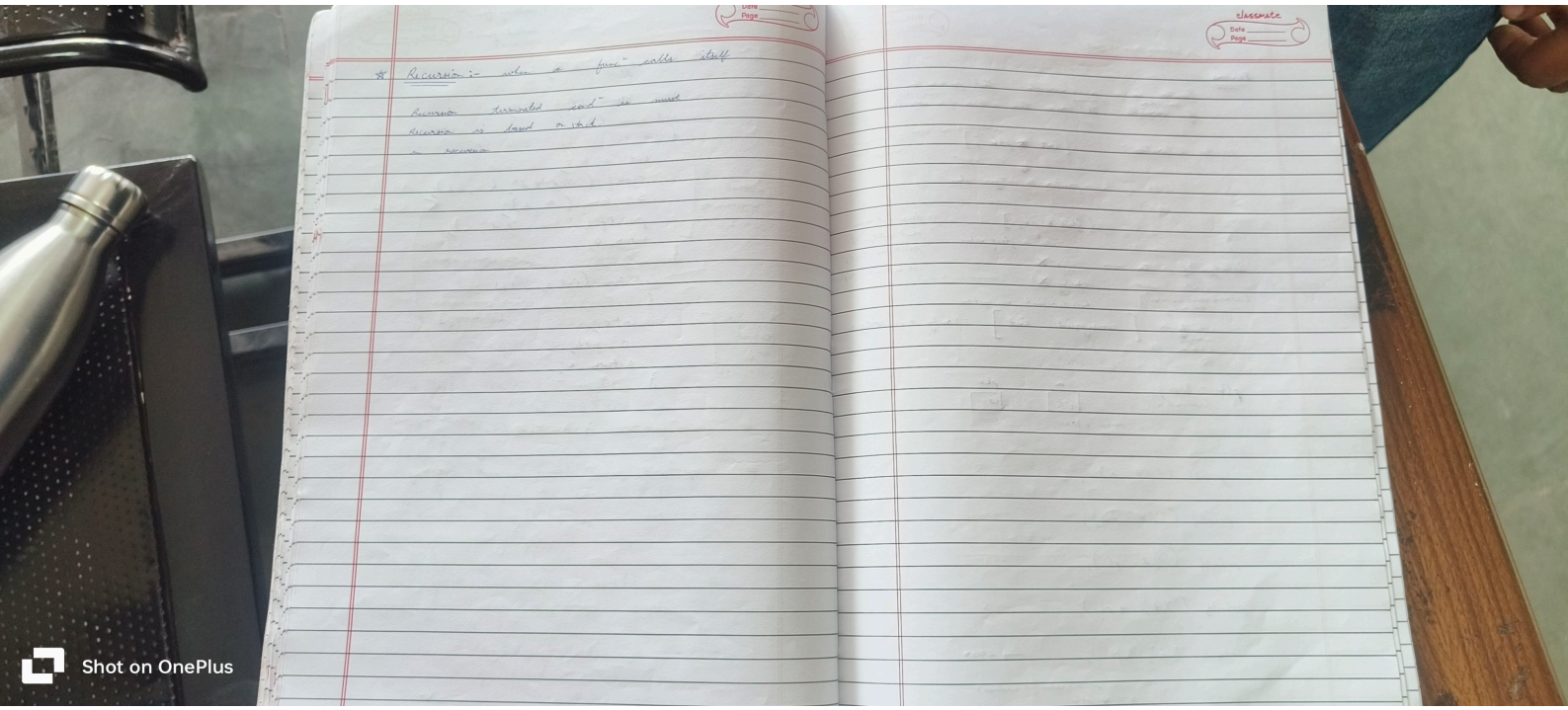
→ new location [photocopy]

formal arg (int a, int b)

ends func^n

func^n factorial → 3 methods with arg res

swap with func^n → no ret with arg


Recursion :- when a func" calls itself
Recursion terminated cond" is must
Recursion is hard to understand
a recursion
Shot on OnePlus

classmate
Date
Page

advantage → fast

Pointers

Pointer is a variable that points to the address of some other variable. That means it stores the address of a variable some other variable.

pointer declaration: [int *ptr;] → not storing value but address

⇓

pointer to integer value

[float *ptr;] → pointer to float value

=> does not mean that datatype of pointer is float, it is pointing to float

datatype of pointer → address → [address can't be -ve or string]

-ve ✗

string ✗ (cannot be 0 also)

hexadecimal no. system

actually datatype is a.k.a. → [unsigned int] => only +ve integers

ptr = &a;
(address of a)

	a		ptr
	+5	=>	1000
storage address	1000		2000

printf ("%d", a); → 5

printf ("%d", &a); → 1000

printf ("%d", ptr); → 1000

printf ("%d", *ptr); → 5

printf ("%d", &ptr); → 2000

a = *ptr → (value at address)

⇓

unary operator

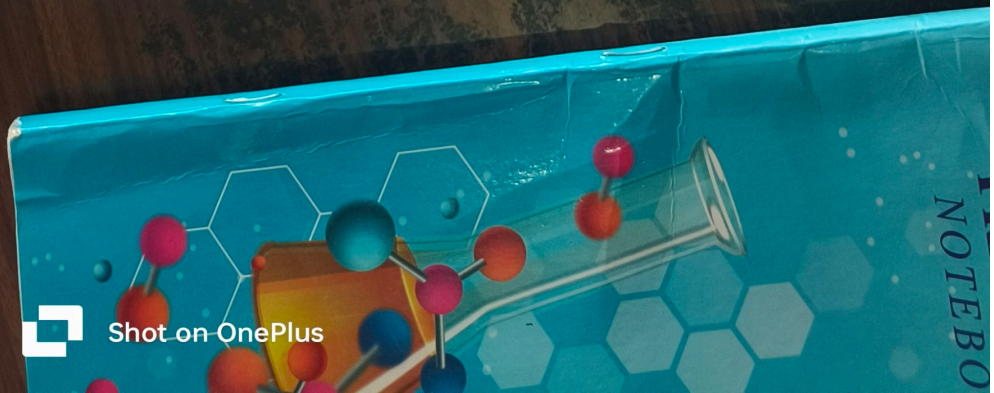
classmate
Date
Page

advantages => fast retrieval, global exchange, consistent data
of pointer

Imp Difference b/w call by value & call by address

Call by value:
value is passed to the func"

Call by value	Call by ref
◦ value is passed to a func"	◦ address is passed to a func"
◦ photocopy of the variables are created.	◦ no photocopy created
☆ ◦ If the formal parameters are changed in the called func", then actual parameters won't get affected.	◦ If the formal parameters are changed then the actual parameters are also affected
◦ slow process	◦ fast process
◦ more memory	◦ less memory
example	example



If write a program of swapping using both call by value & call by reference

Call by value

```
void swap (int, int);
int main ()
{
    int a, b;
    scanf ("%d %d", &a, &b);
    printf ("Before swapping a=%d, b=%d", a, b);
    swap (a, b);
    printf ("After swapping a=%d b=%d", a, b);
    return 0;
}
void swap (int a, int b)
{
    int c;
    c = a;
    a = b;
    b = c;
    printf ("a=%d, b=%d", a, b);
}
```

actual parameter

formal parameter

address

enter 2 numbers : 6 7

o/p =>

	5	10
before		a=5, b=10
photocopy	a=10	b=5
after		a=5, b=10

→ actual parameter not affected

practice previous programs by using func" (factorial, prime, etc)

classmate
Date
Page

call by ref

void swap (int *, int *)
space

swap (&a, &b)

value
void swap (int *a, int *b)
1000 2000
{
int c;
c = *a;
*a = *b;
*b = c;
printf ("a = %d, b = %d", *a, *b);
}

swapping occurs b/w values of 1000 & 2000 inside

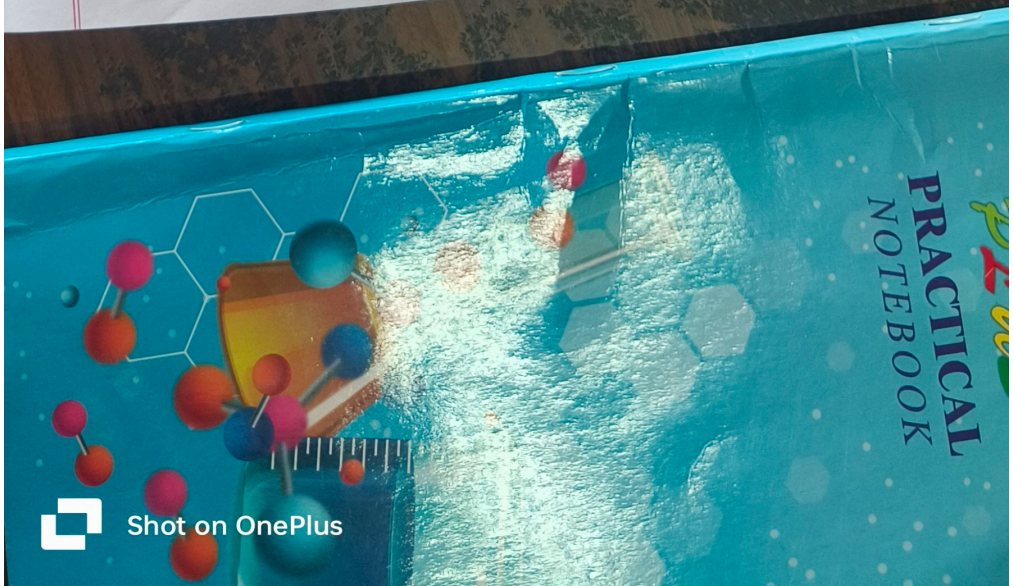
if we write a, b
then O/p = 1000, 2000

O/p =>

5	10
before ...	a=5, b=10
a=10	b=5
after ...	a=10, b=5

=> global change

↳ for file same



```
void modify (int*, int*);
int main()
{
    int a=3, b=4;
    printf("%d %d", a, b);
    modify(&a, &b);
    printf("%d %d", a, b);
}
```

3	4
9	8
3	4

```
void modify(int*, int*)
{
    *a = *a * 3;
    *b = *b * 2;
    printf("%d %d", *a, *b);
}
```

3	4
9	8
9	8

☆ Arithmetic operations that are allowed on pointers

- pointer can be incremented / decr
- add^n of a const. to a pointer
- subtraction of a const from " "
- " " two pointers. ↘ it returns the no. of ind. b/w 2 pointers

not allowed:

- add^n of 2 pointers
- multiplication of a const. to a point.
- division " " " " " "

] may give dangling pointers ⇒ Large system → may give desired ↓ can be avoided

Dangling pointers ⇒ which point to nothing

to print
%u
[%d

you may use:
→ unsigned int
[or vice versa]

classmate
Date
Page

int a = 5;
int *pa;
pa = &a;
a++; → 6
pa++; → 1004 (4 bytes for int storage)

a: 5 (1000)
pa: 1000

immediate next of its data type
pa = pa + 2;
1000 ⇓ → 1008

char ch = 'A';
char *pc;
pc = &c;
c++; → B
pc++ → 1001 (char 1 byte)

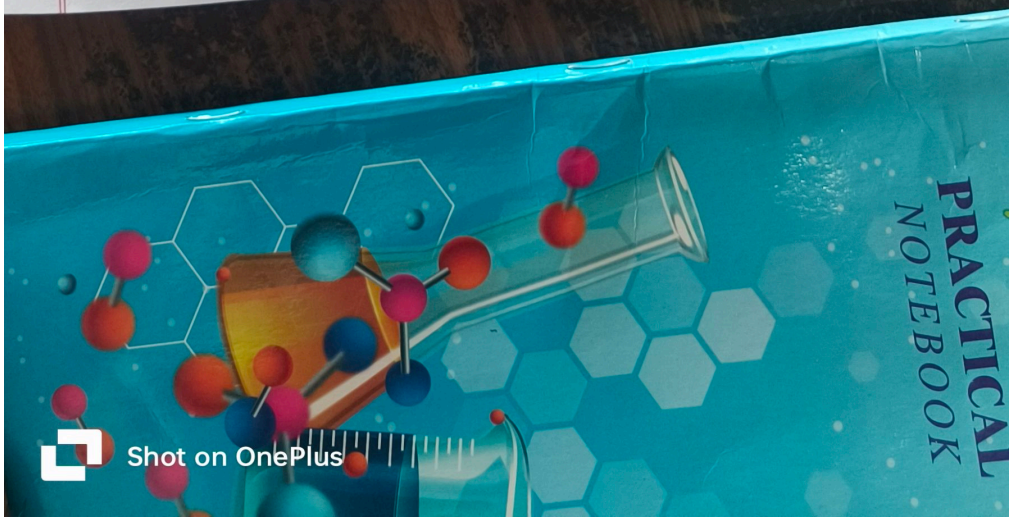
⇒ pc = pc + 1
(add of a const. to a pointer)

* it returns the no. of integers b/w 2 pointer

pa = 1000
pb = 1012
pa - pb = 12/4 = 3 (difference of 3 ints)

pointer to array

1000 1004 1008 1012 1016
int a[5] = {10, 20, 30, 40, 50};
int *pa;
pa = &a; [base address of an array]
printf("%d", *pa); → 10
pa++;
printf("%d", *pa); → 20
pa = pa + 2;
printf("%d", *pa); → 40
pa = pa - 3;
printf("%d", *pa); → 10



pointer to array ⇒ array of integers

Array of Pointers (= array of addresses)

```
int a, b, c, d;
int *p[4] = {&a, &b, &c, &d};
a=10, b=20, c=30, d=40;
for (i=0; i<4; i++)
{
    printf("%d", (*p[i]));
}
```

value

p[i] ⇒ prints address

☆ Strings (array of characters)

how to declare a string

declaration: char a[10]; (10 bytes) ⇒ string contains max 10 characters

only for string

initialize a string: a = "Hello";

'H'	'e'	'l'	'l'	'o'	'\0'				

10 bytes

a[0] = 'H' a[1] = 'e'

'\0' null character [string termination character] automatically placed by the compiler at the end of a string such that no garbage is stored

2D string: row col

char a[5][10];

char a[5][ ] = {"ABC", "DEF", "XYZ", "PQR", "ABCD"}; 5 strings

each string contains max 10 char

a[0] = "ABC"

a[1] = "DEF"

String library functions

```
#include <string.h>
```

① ★ strlen() => to find length of a string

```
#include <string.h>
int main()
{
    int l;
    char a[20];
    scanf("%s", &a);
    gets(a);
    a="Hello";
    l = strlen(a);
    printf("length of string = %d", l);
}
```

String input during run time / compiled time / run time initialization

methods of i/p of string

scanf("%s", &a); → optional as string is itself a pointer

OR

gets(a); → does not accept multiple word string

get string

accepts spaces so it can accept multiple word string → difference

v. string ji ki length nikalni hai

[here l = 5]

★ ② strrev()

```
#
int main()
{
    int l;
    char a[20];
    gets(a);
    strrev(a);
    puts(a);
}
```

Stored as char

scanf ← gets(a);

printf ← puts(a)

→ i/p Hello

→ o/p olleH

check like dictionary

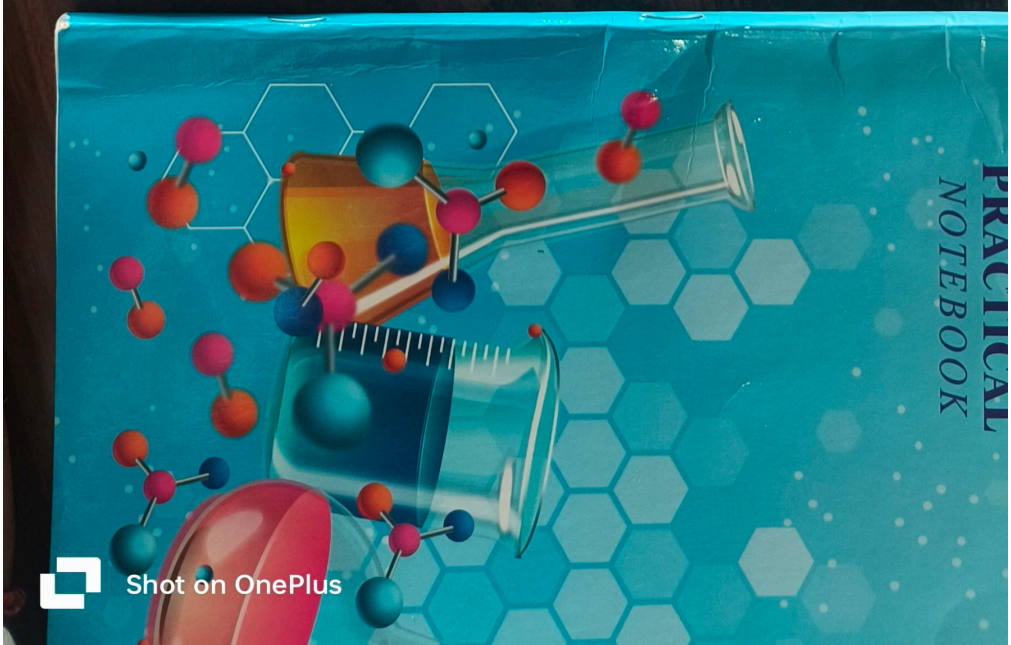
```
l = strcmp ("ABC", "abc");
             65   97      → -ve

l = strcmp ("ABC", "ABCD");
                    more ASCII
                  → -ve
```

III) Write a program to sort 10 strings in alphabetical order.

Solution :-

```
main()
{
 char a[10][20], temp[20];   (10 strings, each can accept 20 chars)
 int i, j;
 for (i=0; i<10; i++)
 {
    gets (a[i]);
 }
 for(i=0; i<10; i++)
 {
  for(j=i+1; j<10; j++)
  {
     b= strcmp(a[i], a[j]);   if((strcmp(a[i], a[j]))>0)
     if (b>0)
     {
      strcpy (temp, a[i]);
      strcpy (a[i], a[j]);
      strcpy (a[j], temp);
      }
    }
 }
  for(i=0; i<10; i++)
  {
     puts(a[i]);
  }
```



classmate
Date
Page

1) To check whether string is palindrome or not.

```
main()
{
    char a[10], b[10];
    gets(a);
    strcpy(b, a);
    strrev(b);
    if((strcmp(a,b)) == 0)
    {
        puts("string is palindrome");
    }
    else
    { puts("not ---");
    }
```

Storage Classes

Structures

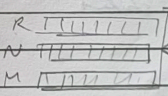
- ~~Also~~ a heterogeneous collection of elements (of different
- * data type). All elements stored in contiguous memory location.
- Structure will be defined & declared separately (array
-
-

define a structure

```
keyword
struct student
{
    int r;          4
    char n[10];     10
    float m;        4
                    --------
                    18 bytes  -> for single student
};  -> imp.         memory reserved
```

Roll Name Marks

10



[array stores 1 column in different memory locations.

Structure stores like: | R_1 | N_1 | M_1 | R_2 | N_2 | M_2 | ...

declaration of structure

total memory = 4x18 = 72

```
struct student s1, s2, s3, s4;
```

depend on syntax/i/p

derived ~~user defined~~	user defined
array	char →
pointer	float → 4
structure	int → 1

Accessing elements of a structure

You can access elements of structure using . (dot) operator

structure variable. structure member = value;

eg: S1.R = 1;

strcpy (S1.N, "ABC") → does the work of S1.N = "ABC" but thing is incorrect

	R	N	M
S1	S1.R	S1.N	S1.M
S2	S2.R	S2.N	S2.M
S3	S3.R		
S4			

```
int main()
{
  struct student
  {
    int r;
    char n[10];
    float m;
  };
  struct student S1, S2, S3, S4;
  S1.r = 1;                          // method 1 (compile time)
  strcpy (S1.n, "ABC");              // method 1 (compile time)
  S1.m = 76.5;                       // method 1 (compile time)
  S2 = { 2, "XYZ", 80.5};            // method 2 (compile time)
  scanf ("%d %s %f", &S3.r, &S3.n, &S3.m);   // methods (run time)
  S4 = S2;                           // you can copy a record
```

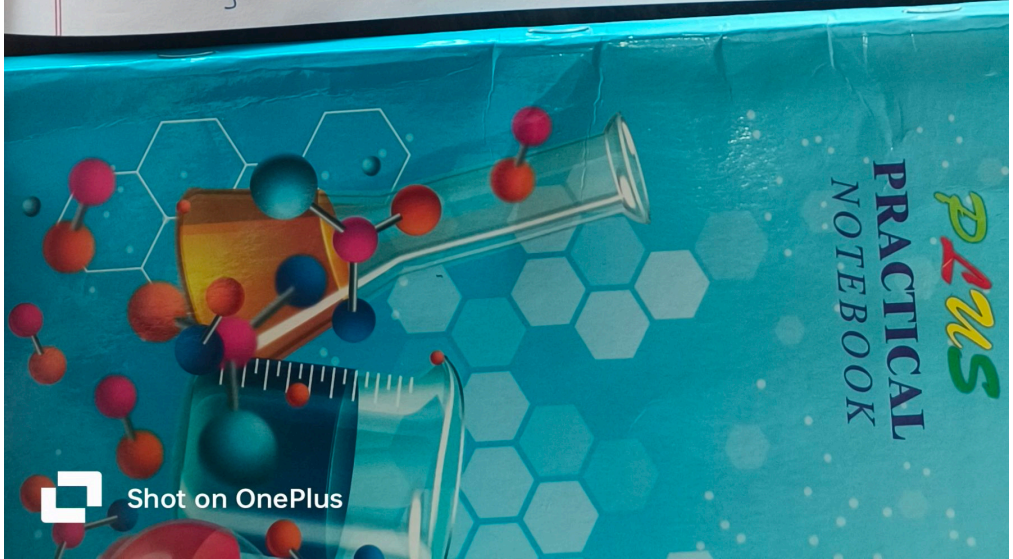
printf ("%d \t %s \t %f \n", s1.r, s1.n, s1.m);
" (" ", s2.r, s2.n, s2.m);
" (" ", s3.r, s3.n, s3.m);
" (" ", s4.r, s4.n, s4.m);

Array of Structure

	R	N	M_1	M_2	M_3	Tot	Avg
10 R.no	S[0] S[1] S[2] S[3]						

```
int main ()
{
  struct student
  {
    int r;
    char n[10];
    float m1, m2, m3, tot, avg;
  };
  struct student s[10];  --> 340 total bytes
  int i;
  for (i=0; i<10; i++)
  {
    printf ("Enter roll number:");
    scanf ("%d", &s[i].r);
    printf ("Enter name:");
    gets (s[i].n);
    printf ("Enter marks of 3 subjects:");
    scanf ("%f %f %f", &s[i].m1, &s[i].m2, &s[i].m3);
    s[i].tot = s[i].m1 + s[i].m2 + s[i].m3;
    s[i].avg = s[i].total / 3;
  }
```

34 each



O/P =>

R	N	Avg

```
for (i=0; i<10; i++)
{
    printf ("%d %s %f \n", s[i].r, s[i].n, s[i].avg);
}
return 0;
}
```

HW

Structure employ

E code	E name	basic salary	HRA	DA	TA	TD

- 5 employs
- HRA = 13% Basic Sala + 14% DA + 5% TA − 10% TD
- Gross salary = Basic + HRA + TA + DA − TD

Nesting of structure

R	N	Address		
	first	surname	HNo	State

Structure within Structure

```
int main()
{
    struct address
    {
        int hno;
        char city[10];
    };
```

```
struct student
{
    int r;
    char n[10];
    struct address a;
} s;
like s1, s2, s3, s4;
```

```
struct employ
{
    int ecode;
    char ename[
    struct addres
} e;
```

s.x = 1;
strcpy (s.n, "ABC");
s.a.hno = 746;
strcpy (s.a.city, "CND");

Passing structure to a func

```
void sort (int [ ]);
int a[10];
sorted  | &a[i];   i/p scanf
        ↓
sort(a);
sorted  ↓          o/p printf
```

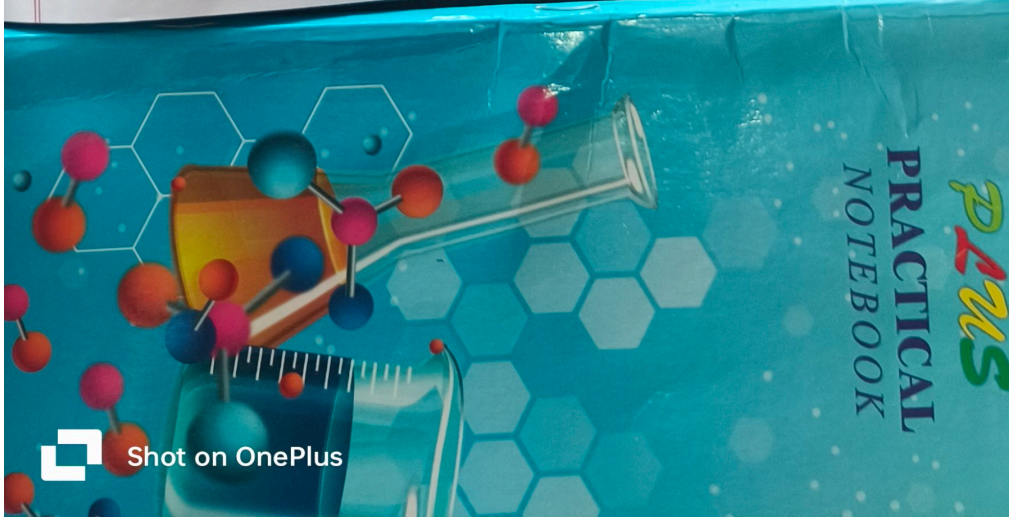
```
void sort int(b[ ])
{
  | sorting loop
}
```

2 methods:
(a) element by element
(b) whole structure

→ (a) write this example in exam

```
void copy (int, char, float);
main()
{
struct student
{
  int x;
  char n[10];
  float m;
} s;
scanf ("%d %s %f", &s.x, &s.n, &s.m);
copy (s.x, s.n, s.m);
}
```

```
void copy (int a, char b[],
           float c)
{
printf ("%d is %f", a,b,c);
}
```
call by value



(b) whole structure

```
struct student
{
    int r;
    char n[10];
    float m;
};
void copy(struct student);
int main()
{
    struct student s;
    scanf("%d %s %f", &s.r, &s.n, &s.m);
    copy(s);
}
void copy(struct student s1)
{
    printf("%d %s %f", s1.r, s1.n, s1.m);
}
```

defined outside (global)

can't be done in (a)

Pointer to structure

```
int main()
{
    struct student
    {
        int r;
        char n[10];
        float m;
    };
```

~~scanf("%d %s %f", &s.r, &s.n, &s.m);~~

→ dereferencing operator

```
struct student s;
struct student *ps;
ps = &s;
scanf ("%d %s %f", &s.r, &s.n, &s.m);
printf ("%d %s %f", s.r, s.n, s.m);
printf ("%d %s %f", ps->r, ps->n, ps->m);
}
```

↳ works like '*' but for structures

PYQs

Difference b/w array & struct
" " structure & union
" " automatic & static

Diff

Union

like structures, unions are also heterogeneous collection of elements

difference b/w union & struct

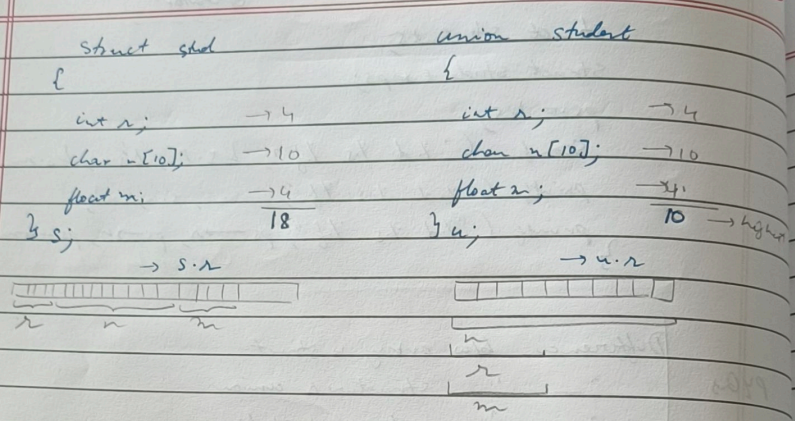
- structures can be defined using struct keyword
- " " " union "

• all the members of a structure occupy different memory location separately.	• in unions, all members share the same storage space. memory allocated to union is the highest data type memory.
• memory allocated = sum of all the members' data type memory.	• only one member can be used at one time due to space sharing.
• software	• hardware
	• used to conserve memory

union → generally used in hardware

⇓

used to conserve memory



typedef

it is a user defined datatype which is used to create a new datatype. It does not create a new datatype but it merely adds a new name to some existing datatype.

typedef existing datatype new data type

```
int a,b;
typedef unsigned int ui;
ui a,b;
```

shorter/easy way

Bitfield

- also used to conserve memory.
- they are the must be members of structures.
- you cannot create arrays of bitfield
- if you want to store any value/data in bits rather than bytes, you can use bitfields.

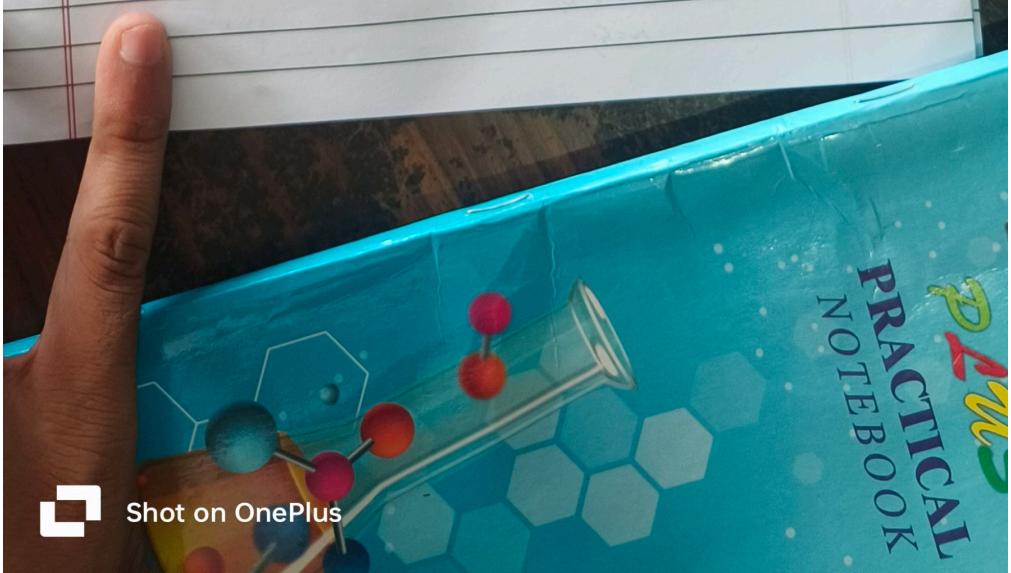
4 bytes → 32 bits

classmate
Date
Page

declaration — always declared by unsigned int

```
struct s
{
    unsigned int a : 3;
}
```

a → 3 bits



C Preprocessor

converts source code to expanded source code

4 preprocessor directives → always start with a # symbol (viva question)

1. File inclusion (#include)
2. MACROS (#define)
3. Conditional Compilation (#ifdef, ifndef, #undef, #if, #endif)
4. miscellaneous directive (#pragma) → not in syllabus

1. FILE INCLUSION

→ #include <stdio.h> → pre made files in C
#include <math.h>

→ #include "myfile.h" → your own file
(it includes user created files)

2. Macros

- used to define symbolic constts.
- can be used in place of functions
- can replace conditions
- multiple replacements → jo replace karega

#define MACRO TEMPLATE — MACRO EXPANSION → jo replace hoga

eg: #define PI 3.1415

```
int main()
{
    int a, r;
    r = 5;
```

difference b/w macros & functions

classmate
Date
Page

→ doesn't use memory unlike variable 3.14.

```
a = PI * r * r;
printf("%d", a);
}
```

⇒ in matrix programs, for(i=0; i<3 --- > is used multiple times

```
# define UPPER 3
;
for (i=0; i<UPPER; i++)
```

— — —

— — UPPER

```
# define INRANGE ((a>b) && (a<60))
```

grading prog. ⇐

```
int main()
if ((a>40) && (a<60)) ——can be replaced by——> if (INRANGE)
```

```
# define AND &&
if((a>b && (a>c)) ——can be replaced——> if((a>b) AND (a>c))
```

★ MACROS with arguments

```
# define AREA(r) (3.14*r*r)
int main()
{
    int a, r=5;
    a = Area(r);
    printf( , a);
}
```

control goes back to back

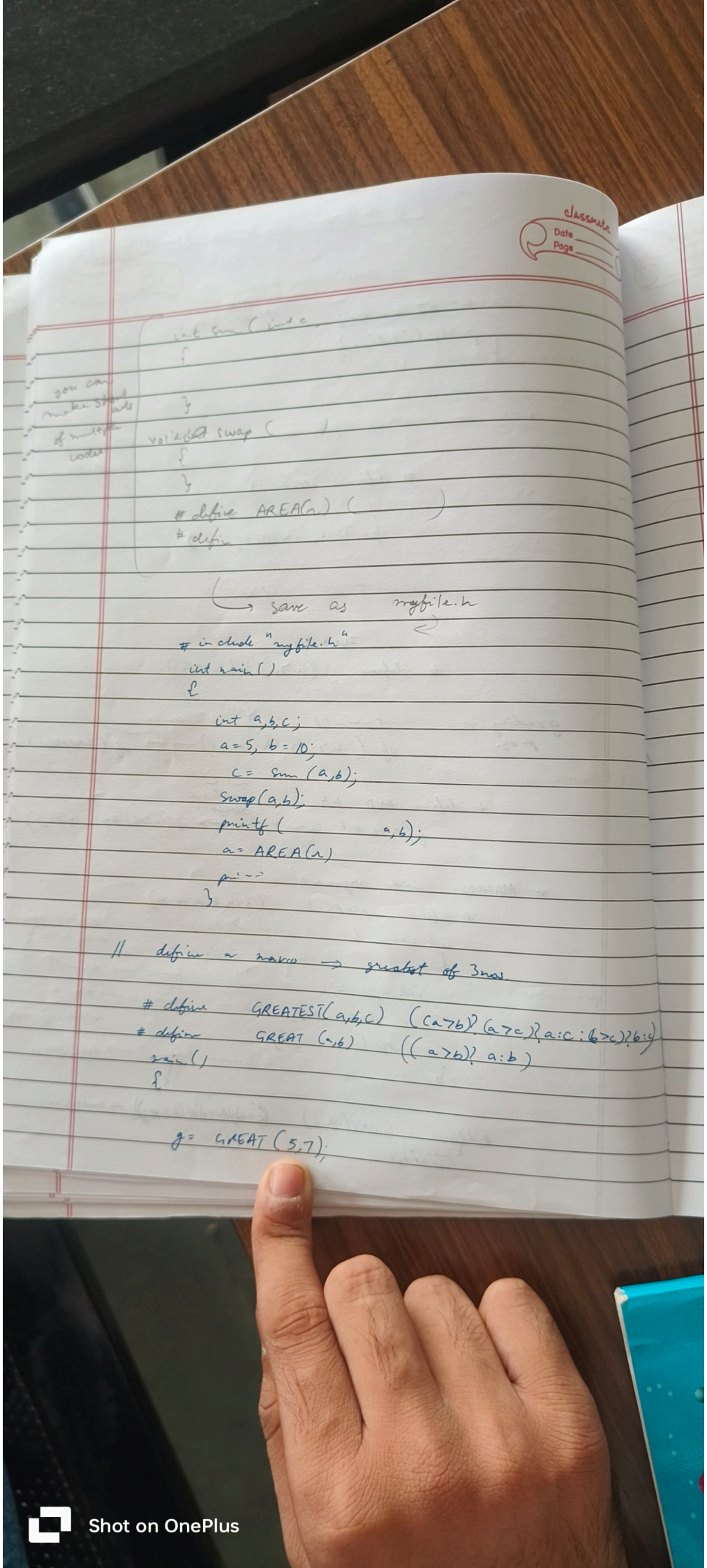
func^n → time ↑

macros → take ↓ like cars park

func^n → less space

```
# define SWAP(a,b) (c=a; a=b; b=c;) → multiline
{int c;
SWAP a,b}
```

PRACTICAL NOTEBOOK


classmate
Date
Page
int sum (int a
{
}
void swap ( )
{
}
# define AREA(r) ( )
# defin
you can make short cuts of multiple codes
save as myfile.h
include "myfile.h"
int main ( )
{
int a,b,c;
a=5, b=10;
c= sum (a,b);
swap(a,b);
printf (a,b);
a= AREA(r)
pi~>
}
// define a macro → greatest of 3nos.
define GREATEST(a,b,c) (a>b)?(a>c)?a:c:(b>c)?b:c
define GREAT (a,b) ((a>b)? a:b)
main ( )
{
g= GREAT (5,7);

③ Conditional Compilation

```
#define TEST
int main
{
    statement 1
    St 2;
#ifdef TEST
    st 3;
    st 4;
#endif
    st 5;
```

#undef TEST → not defined

only works if MACRO is defined

cond^n only on these

#ifndef => if not defined