Introduction to Programming... Unit-I

- What is Problem Solving?

⇒ A plethora of real life problems in digital world need to be solved through programming.

→ 1. Problem-solving is a crucial skill in computer science and programming. It involves breaking down complex tasks into smaller, manageable steps and creating a clear plan to solve them.

2. Algorithms, flowchart and pseudocodes are powerful tools that aid in this process.

- Steps in Problem Solving :-

⇒ 1. Understand the Problem :-

⇒ • Clearly define the problem you need to solve.

• Identify the input data, constraints and desired output

2. Plan the solution :-

⇒ • Break down the problem into smaller sub-problems or steps.

• Decide on the appropriate algorithmic approach (e.g. → searching, sorting, recursion).

3. Design the algorithm :-

⇒ • Create a step-by-step procedure to solve each sub-problem.

• Use flowchart symbols to represent each step visually.

4. Implement the Algorithm :-

=> • Translate the algorithm into a specific programming language.

• Write code following the algorithm's step.

5. Test and Debug :-

=> • Test the program with various inputs, including edge cases.

• Debug and correct any errors or unexpected behaviour.

6. Optimize and Refine :-

=> • Analyze the efficiency of the algorithm.

• Look for opportunities to improve performance, if needed.

Algorithm...

=> An algorithm is a step-by-step procedure or set of instructions to solve a specific problem. It serves as a blue print for solving problems and can be implemented in various programming language.

• Algorithms are independent of programming languages.

• They consist of well-defined steps that must be followed in a specific order.

• Good algorithms are efficient, accurate and produce correct results for all valid inputs.

* Algorithm to find factorial of a number :-

⇒ Step 1 :- Start

Step 2 :- Read the input number from the user.

Step 3 :- Declare the initialize variable fact = 1 and i = 1.

Step 4 :- Repeat the loop until i <= num.

Step 4.1 :- Fact = fact * i

Step 4.2 :- i = i + 1

Step 5 :- Print fact to get the factorial of a given number.

Step 6 :- Stop.

★ Example :-

→ Radius (input);

⇒ Step 1 :- Start

Step 2 :- Read the value of radius r from the user.

Step 3 :- Take two variable 'c' for circumference and 'a' for area.

Step 4 :- Calculate $2\left(\frac{22}{7}\right) * r$ and store the Calculated value in C.

Step 5 :- $(22/7) * r * r$ and store the calculate value in a.

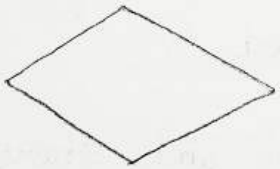
Step 6 :- Print the value C and a.

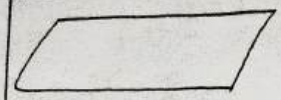
Step 7 :- Stop

Flowchart...

=> A flowchart is a visual representation of an algorithm using symbols and arrows to depict the sequence of steps in a process. It helps in understanding the logic and structure of the algorithm before actual coding.

- Flowcharts are use standard symbols to represent different actions, decisions, inputs and outputs.
- Symbols include rectangles (process), diamonds (decision), arrows (flow of control) and parallelograms (input/output).
- Flowcharts improve communication between developers and stakeholders, aiding in debugging and refining algorithms.

Name	Symbol	Usage
Start / Stop		Used only at the beginning and end of the flowchart.
Process		Used to 'do something' for example, assign a variable a value, or carry out a calculation.
Decision		Used to change the direction of a program. This is the only block that can have more than 1 arrow coming out of it, and any arrows should be labelled with possible answers to the question posed.

Input / output		Used for a user input or to output or print something.
Connector		If there is not enough space to fit the flowchart on one page, connectors can be used to show where the flowcharts should be joined together.
Direction of flow		Arrows show how to travel through the flowchart.

→ Rules for creating flowchart :-

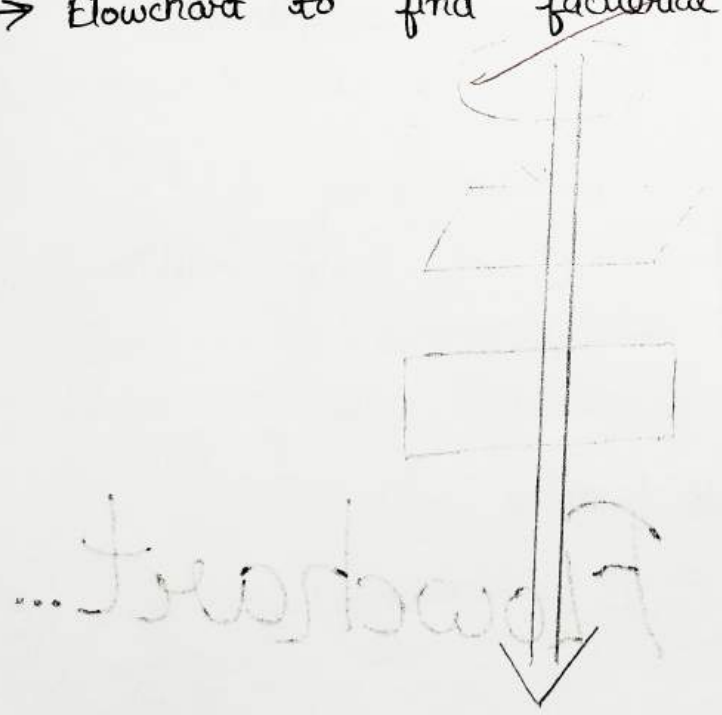
=> 1. Flowchart opening statement must be 'start' keyword.

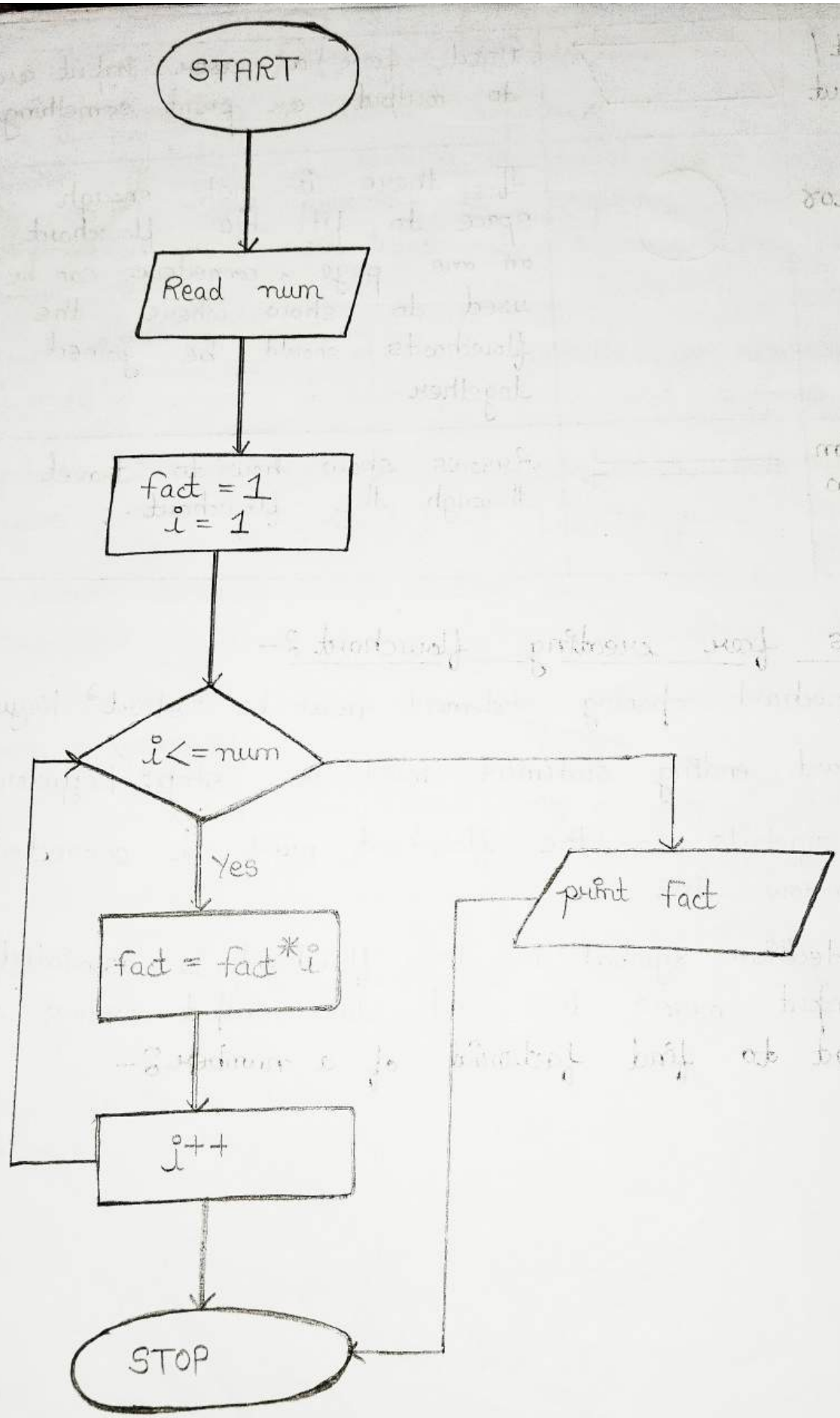
2. Flowchart ending statement must be 'stop' keyword.

3. All symbols in the flowchart must be connected with an arrow line.

4. The decision symbol in the flowchart is associated with one input arrow line and two output arrow lines.

→ Flowchart to find factorial of a number :-





Flowchart...

* Programming for Problem Solving using C :-

- Language C :-

→ Middle level language
→ Structural language → Functions, code blocks
→ Procedure oriented

* Basic Structure unit ⇒ The smallest unit in C program is tokens.

1) Keywords (int, auto, break, continue)
2) Identifiers
3) operators
4) Strings ('A', '$', 'Hello')
5) Constants
6) Special Symbols ⇒ ( )

* Rules :-

⇒ Identifiers are used for naming program elements, like variable, function, arrays etc.

- Rules to be followed while naming an identifier :-

i) An identifier must begin with a letter followed by the letters or digits.

ii) '_' is permitted and considered as a letter. So an identifier can also start with an '_'.

iii) Name of an identifier cannot be anyone out of the reserved words / Keywords.

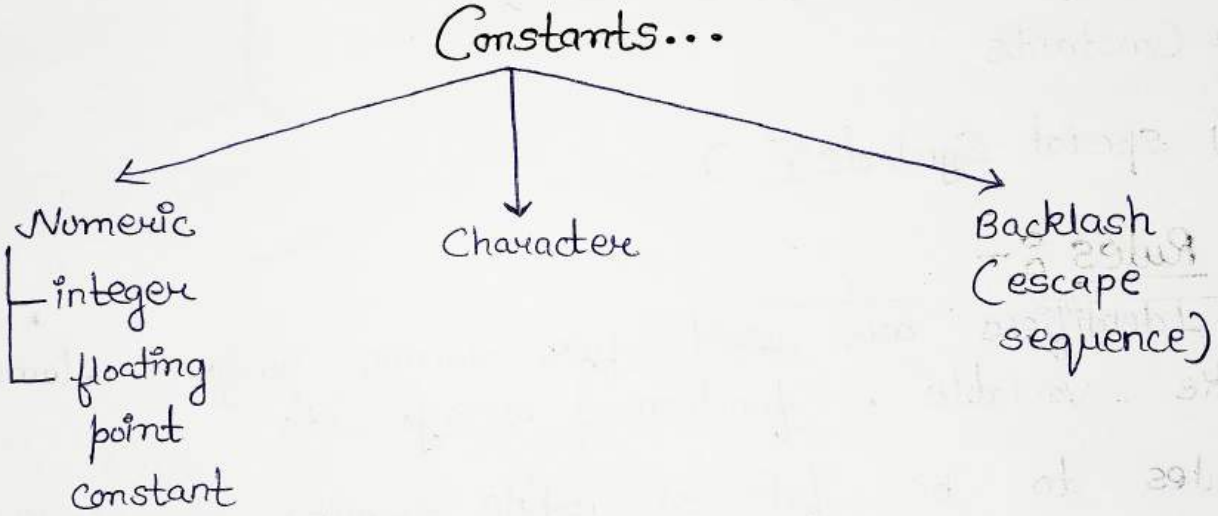
iv) C language is case sensitive, if you are following

a particular case for the identifier name, it should be followed as it is through out the program.

Sum, SUM, sum.

→ Variable names are generally written using lower case letter and upper case letters are used for symbolic Constants.

Imp.
ANSI => (American National standard institute) 31 characters are permitted to be taken in an identifier name but it is a good practice to use fewer character upto 8 characters.



→ Const int height = 12;

→ Const float height = 12.7;

→ Const char height = 'A';

→ Const * char S = "Hello";

• Two ways to declare constant :—

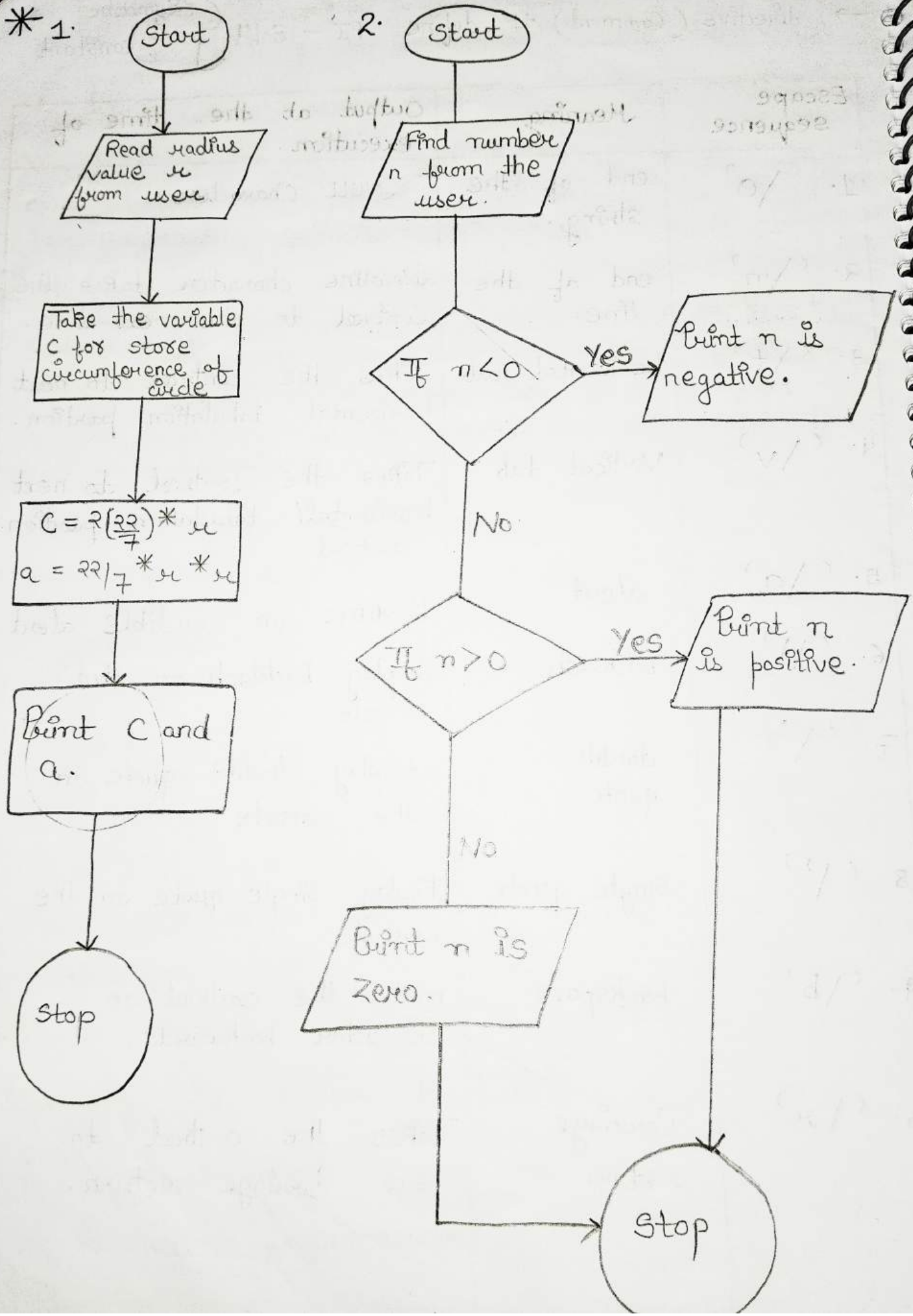
1) L Const Keyword.

2) L # define preprocessor.

→ preprocessor # include <stdio.h> } Symbolic constant
→ directive (Comment) # define PI - 3.14

Escape sequence	Meaning	Output at the time of execution.
1. '\0'	end of the string.	NULL Character
2. '\n'	end of the line.	Newline character takes the Control to the next line.
3. '\t'	horizontal tab	Takes the control to next horizontal tabulation portion.
4. '\v'	Vertical tab	Takes the control to next horizontal/ vertical tabulation portion.
5. '\a'	alert	Produces an audible alert.
6. '\\'	backlash	display backlash on the console.
7. '\"'	double quote	display double quote on the console.
8. '\''	single quote	display single quote on the console.
9. '\b'	backspace	moves the control on character backwards.
10. '\r'	Carriage return	Takes the control to next carriage return.

Flowchart ...


* 1.
Start
Read radius value r from user
Take the variable C for store circumference of circle
C = 2(22/7) * r
a = 22/7 * r * r
Print C and a.
Stop
2.
Start
Find number n from the user.
If n<0
Yes
Print n is negative.
No
If n>0
Yes
Print n is positive.
No
Print n is zero.
Stop

Pseudocode :-

(A high-level algorithmic description)

=> Pseudocode is a simple and informal way to describe the logic of an algorithm without getting into the specifics of programming syntax. It bridges the gap between the algorithm's high-level logic and its actual implementation in a programming language.

- Pseudocodes for finding Factorial of Number :-

=> Read number

Fact = 1

i = 1

WHILE i <= number

Fact = Fact * i

ENDWHILE

WRITE FACT

- Another example :-

=> Start

Read n

sum = 0

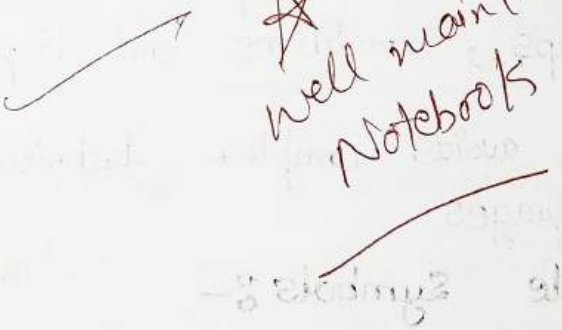
for i = 1 to n

do sum = sum + i

end for

Display sum

Stop



* Key features of pseudocodes :-

=> 1. Language independent :-

=> • Pseudocodes is not tied to any specific programming language.

• It focuses on expressing the algorithm's logic and steps in a human - readable format.

2. Readable and understandable :-

=> • Pseudocode is designed to be easily comprehensible by both technical and non- technical individuals.

• It helps convey the algorithm's flow and structure clearly.

3. No Strict Syntax :-

=> • Pseudocode does not have a strict syntax like programming languages.

• It uses natural language and simple symbols to represent actions and decisions.

4. Uses natural language :-

=> • Pseudocode employs english- like phrases to explain steps, conditions and loops.

• It avoids complex technical jargon found in programming languages.

5. Simple Symbols :-

=> • Pseudocodes uses symbols like arrows, identation, and labels to depict the flow of algorithm.

• It uses assigment statements and conditionals to describe actions and decisions.

• Benefits of pseudocodes :-

=> 1. Planning and design :-

=> • Pseudocodes aids in designing and planning algorithms before actual coding.

• It helps in identifying logical errors and refining the algorithm's logic.

2. Collaboration and Communication :-

=> • Pseudocode is easy to understand and can be used to explain algorithms to team member or stake-holders.

3. Transition to coding :-

=> • Developers can easily translate pseudocode into code in a specific programming language.

4. Focus on logic :-

=> • Pseudocode shifts the focus from syntax to algorithmic logic, making it easier to concentrate on problem-solving.

5. Testing and Validation :-

=> • Algorithm described in pseudocode can be validated for correctness before coding begins.

* Algorithm to find minimum 3 numbers :-

=> Step 1 :- Read three numbers : num 1, num 2, num 3.

Step 2 :- Initialize a variable min-num with the value of num 1.

Step 3 :- If num 3 < min_num, set min_num = num 3

Step 4 :- If num 3 < min_num, set min_num = num 3.

Step 5 :- Display min_num as the smallest of the three numbers.

* Algorithm to check number is even or odd :-

⇒ Step 1 :- Read the input number.

Step 2 :- Calculate the remainder of the number when divided by 2.

Step 3 :- If the remainder is 0, then the number is even.

Step 4 :- If the remainder is 1, then the number is odd.

Step 5 :- Display the result.

→ pseudocode ...

⇒ READ number

remainder = number % 2

If remainder == 0

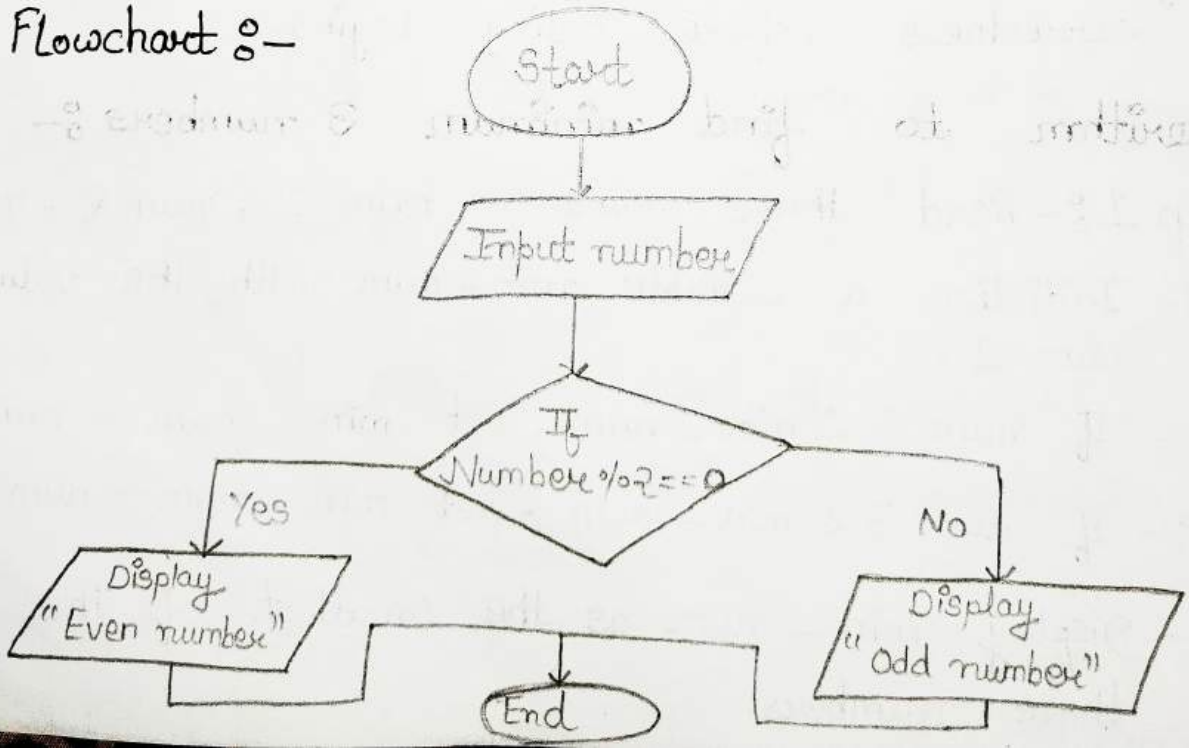
WRITE "Even Number"

ELSE

WRITE "Odd Number"

ENDIF

→ Flowchart :-



Imp.

★ Variable or Declaration of a variable :-

=> 1. [storage class] 2. [type qualifier] 3. [type modifier (= value)] [datatype]

- storage class → auto or register storage class
- type qualifier → Constant
- type modifier (= value) → unsigned and signed
- datatype → int x=100

1: Storage class :-

=> The mode in which the variable is assinged memory space is determined by its storage class. C.P.U registers and main memory locations are the two kinds of memory spaces. Here variable value can be stored. There are variable value are of four types of storage classes.

i) Automatic

ii) Register

iii) External

iv) Static

→ Storage class of a variable determines :-

i) The default initial value of a variable.

ii) Scope of the variable.

iii) It tells life of the variable.

iv) Weather storage location is memory or C.P.U register.

1: Auto storage class :-

=> Any variable declared as auto takes a default initial values as garbage value. This variable is treated as a local variable. The scope of the variable lies within the function in which it is define.

2. Static Variable :-

=> The variables declared as static variable take the default initial value as zero. These variables are also treated as local variables. The value of the variable persist b/w different function calls.

3. Register :-

=> The variable declared as register store their value in C.P.U registers unlike the variables declared as auto extern or static. The default initial value of a register variable is taken as garbage value. These variables are treated as local variables.

4. Extern :-

=> The variable declared with extern keywords are treated as global variable and treated variable. The default initial value for these variable is zero.

Assigment → 1

1. Differentiate between a flowchart and an algorithm using an example.

=> Algorithms :-

=> An algorithm is step-by-step method to solve problems. It includes a series of rules or instructions in which the program will be executed.

Example → Convert temp. from Fahrenheit (°F) to Celsius (°C); using algorithm :-

Step 1 :- Read temp. in Fahrenheit.

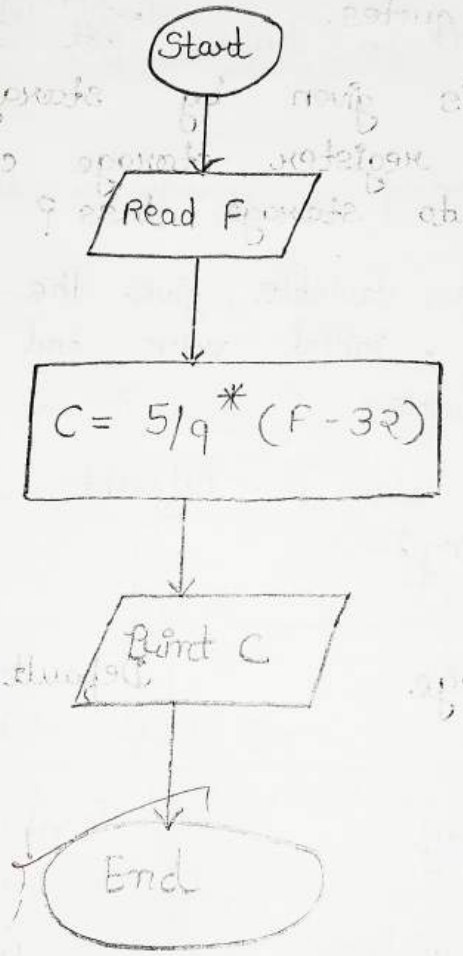
Step 2 :- Calculate temp. with formula $C = \frac{5}{9} * (F - 32)$

Step 3:- Print C.

⇒ Flowchart :-

⇒ A flowchart is a pictorial representation of an algorithm. It uses different patterns to illustrate the operations and processes in a program.

Example → Convert temp. from Fahrenheit (°F) to Celsius (C°); Using flowchart :-



2. What do you mean by a variable in C and what are different kinds of data types that can be used to declare a variable?

⇒ In C, variables are containers for strong data values, like numbers and character.

→ There are different data types that can declare a variable :-

① int :- stores integers (whole no.), without decimals such as 123 or -123.

② float :- stores floating point numbers with decimals, such as 19.99 or -19.99.

③ Char :- stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes.

Mention user defined & derived types also

3. What information is given by storage class of a variable ? How register storage class is different from auto storage class ?

⇒ Storage class of a variable gives the determination of visibility, lifetime, initial value and memory location of any given variable.

→ Register storage class is different from auto storage class from following :-

Storage class	Storage	Default initial value
Auto	Memory	Unpredictable
Register	CPU Registers	Garbage

4. What are pseudocodes ? Write pseudocode to Calculate and display the simple Interest.

⇒ Pseudocode is an informal way of writing a program for better human understanding. It is written in simple English, making the complex program easier

to understand pseudocode cannot be Compiled or interpreted.

→ Pseudocode to calculate simple interest :-

⇒ Input :- Principal p, interest rate r, time t.

Output :- Simple Interest (p, r, t);

Si → (p * r * t) / 100

Return Si

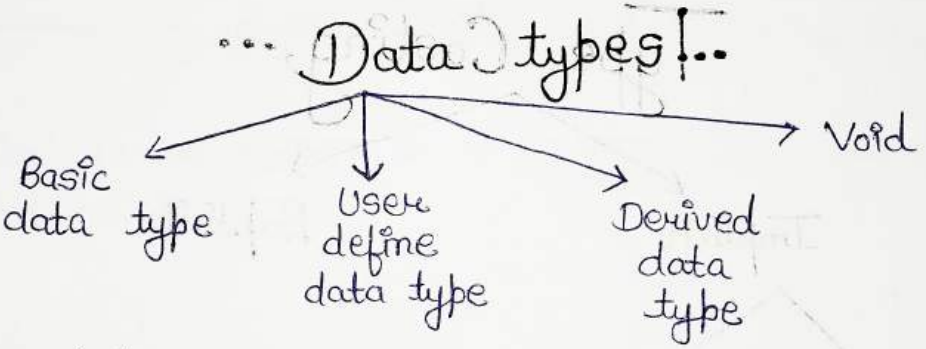
End Procedure

5. What will be the output of the following program C.

⇒ Output :- 5

6. How the output is change if print statement is changed as follows :-

⇒ Output :- ~~5.75~~ 0.00 (logical error).



→ Enumerated data types :-

⇒ enum month { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

enum month { Jan, Feb, march, - - - - -, Dec };

enum shapes { triangle = 3, quadrilateral = 4, pentagon = 5 };

→ enum month

m1, m2;

m1 = June;

m2 = Jan;

Print (" difference b/w m1 and m2 is o/d", m1 - m2);

→ enum shapes s1;
s1 = pentagon;

printf("Number of sides in SI are %d", s1); ⑤

★ **Structure / Unions :-**

⇒ printf("%p", &x);

→ l-value (location value) } of a variable.
→ r-value (read value) }

int x; (2 bytes are allocated)
int x = 10;

int x; }
x = 10; }

extern int x ; // memory is not allocated.

Type Casting...

- Implicit
  - Promotion
  - Transcation ⇒ ~~float~~ x = 73.159
- Explicit

→ char ch = 'B';
int x ;
x = ch; // Implicit type Casting or Promotion
printf("%d", x);

int y ;
y = x; // Implicit type Casting or promotion
printf("%d", y);

→ Another example :-

⇒ i) float z ;
int x , y ;
y = 3 ;
float z = x/y ;
printf (" %d divided by %d gives result %f," x,y,z);
70 ——— 3 gives result 23.0.

ii) float z ;
int x , y ;
x = 70 ;
y = 3 ;
(float) x/y ;
printf (" %d divided by %d gives result %f," x,y,z);
70 ——— 3 gives result 23.333.

→ # include <stdio.h>
int main ( )
{
char ch ;
ch = 291 ;
i) printf (" %d %c", ch , ch);

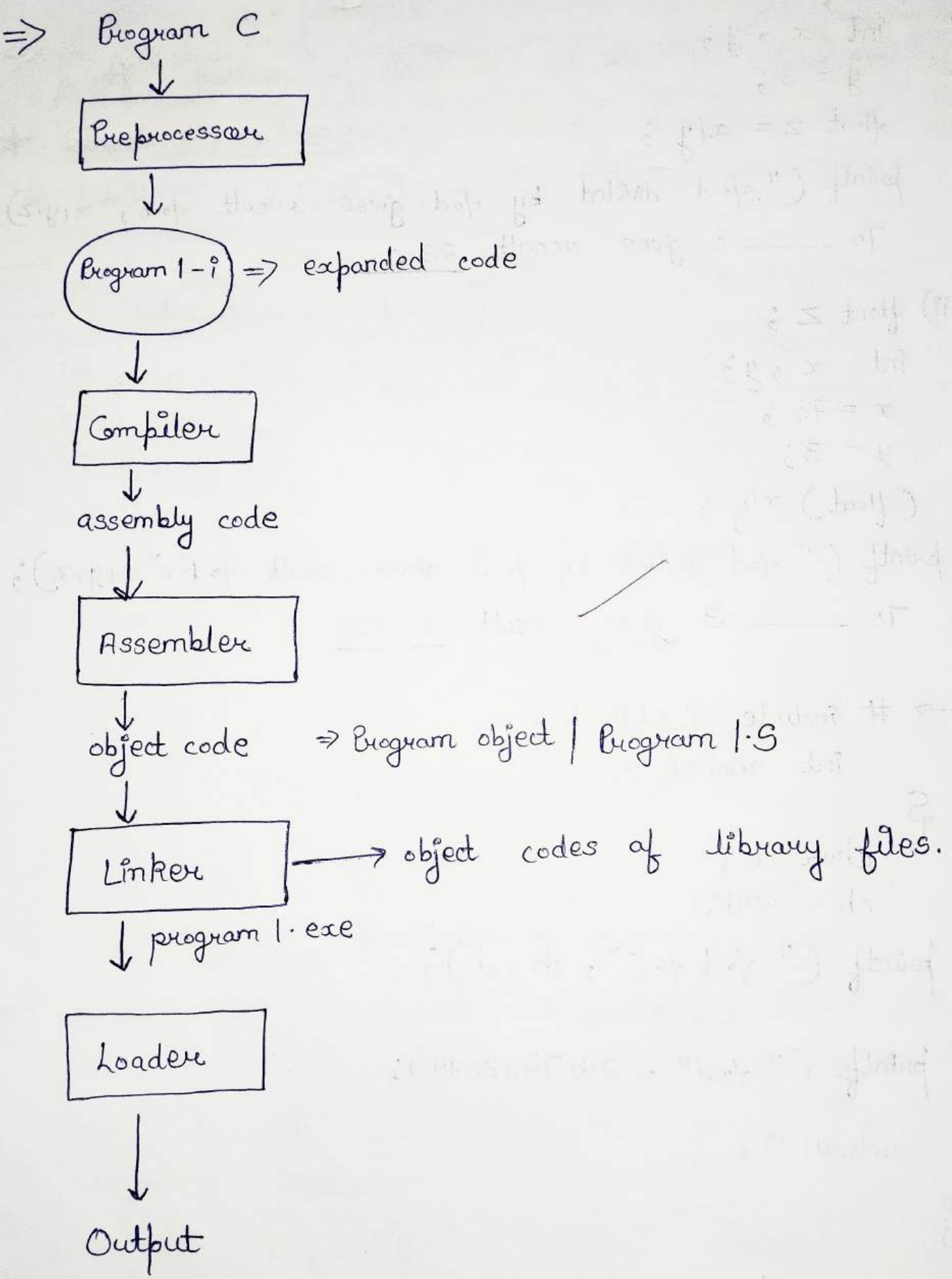
ii) printf (" %d", 2147483649);

return 0 ;
}

Ans → i) 35#

Ans → ii) - 2147483647

- **Compilation Process :-**



Operators...

⇒ An operator is a character that represents a specific mathematical or logical action or process.

Types of operators

1. Unary operator :-

⇒ The operator that act upon just a single operand for producing a new value. Their associativity is from right to left.

Ex → ++ , --

++a ⇒ Preincrement

a++ ⇒ Postincrement

--a ⇒ predecrement

a-- ⇒ postdecrement

2. Arithmetic operator :-

⇒ This operator are used to perform mathematical operations in a C program. They can be used in programs to define expressions and mathematical formulaes.

Ex → + , - , * , / , %.

3. Relational operator :-

⇒ It is used to compare two values in C language. If relation is true, it returns 1. However if the relation is false, it returns 0.

Ex → < , <= , > , >= , == , !=

== ↓ equality

!= ↓ non-equality

4. Logical operator :-

=> It is used to combine multiple conditions / constraints. It returns either 0 or 1, it depends on whether the expression result is true or false.

Ex → &&, ||, !, ¦¦.

5. Bitwise operator :-

=> It is a type of operator that operates on bit arrays, bit strings, and tweaking binary values with individual bits at the bit level.

Ex → &, ¦, ~, ^, <<, >>

- & → AND
- ¦ → or
- ~ → not
- << → left shift
- >> → right shift

6. Assigment operator :-

=> It assigns the value of the right-hand operand to the storage location named by the left-hand operand.

Ex → =, +=, -=, *=, /=, %=

7. Ternary operator :- OR Conditional operator :-

=> The programmers utilize the ternary operator in case of decision making when longer conditional statements like if and else exist. When we use an operator on three variables or operands.

Ex → ? , :

→ If bits are same, resultant will be zero

0 0 = 0

1 1 = 1

→ If bits are different, resultant will be one.

0 1 = 1

1 0 = 1

* int n = 10;
  m = -3;
  int x = n--;

  x = n } 10
  n = n-1 } 9

  int y = m++;

  y = -3

  m = 2

---

* Void main()
  {
  int j = -10;
  int x = --j;
  printf("%d%d%d%d", (j-x++ + (++x), j, x);
  -11 -10 -10 -11 -9
  => -31

---

* Void main()
  {
  int j = -8;
  int x = --j;
  printf("%d%d%d%d", (x + j++) + x--, j, x)
  x = -10 -10 -9 -9 -8 -10

  j = -9

  => -28 Answer

*→ Void main( )

{

int a = 4;

y = 1;

int c = 3;

int x = 4 * a * y/c - a * y/c;

printf ("%d", x);

}

$4 * 4 * 1/3 - 4 * 1/3$

$16 \times 1/3 = 4 * \frac{1}{3}$

$16/3 - 4 * \frac{1}{3}$

$16/3 - 4/3$

$12/3$

4 Ans

---

*→ Void main( )

{

float a = .17;

float b = 3;

print ("%d", a%b);

}

ERROR

↓

(% does not work on float).

* → Void main()

{

int a = -10;

b = 3;

c = 9;

int avg = $\frac{a+b+c}{3}$;

printf("%d", avg);

}

⇒ -10+3+9/3

⇒ -10+3+3 = -4

Errors...

⇒ Errors are the problems or the faults that occur in the program, which makes the behaviour of the program abnormal. Programming errors are also known as the bugs or faults, and the process of removing these bugs is known as debugging.

Types of errors...

1. Syntax errors :-

⇒ It is also known as Compilation errors are thrown by the compiler and occur at the time of Compilation. These errors occur due to the mistakes while typing or when the syntax of specified programming language are not followed.

→ missing ⇒ ; , } , " , int x;

↓

syntax error

2. Runtime errors :-

=> Errors that occur during the execution of a program are called runtime errors. When a program is running and it is not able to perform any particular operation it means we have encountered a runtime error. For example :- If we try to input a negative number to finds its square root. It will produce a runtime error.

(If we try to divide a number by zero). ★

↓↓

example

3. Logical errors :-

=> These errors may occur when the programmer writes a wrong logic for the program. For e.g. → Programmer allows any no. apart from 1 to 12 to be taken as value for a month variable and performs calculations according to this invalid value.

```
=> # include <stdio.h>
   int main( )
{
   int n1 , n2;
printf ("Enter value of n1 and n2");
scanf ("%d%d , &n1 , &n2);
if (n2=0)
{ printf (" Division not possible"); }
else
  printf ("%d", n1/n2);
  return 0;
}
```

4. Linker error :-

=> These errors occur during the linking process. For example :- If we try to call a function which is not define then linking error not be displayed.

5. Semantic error :-

=> Errors that occur because the compiler is unable to understand the written code are semantic errors. Although the code may be syntactically correct but it is making no sense to the compiler.

Example :- a + b = c ;

★ Decision control statements / Conditional statement :-

=> → if
→ if else
→ Nested if
→ Nested of else

```
if (Condition)
    Statement ;        → single statement

if (Condition)         → (multi statement)
{
    set of statement
}
```

* Syntactically :-

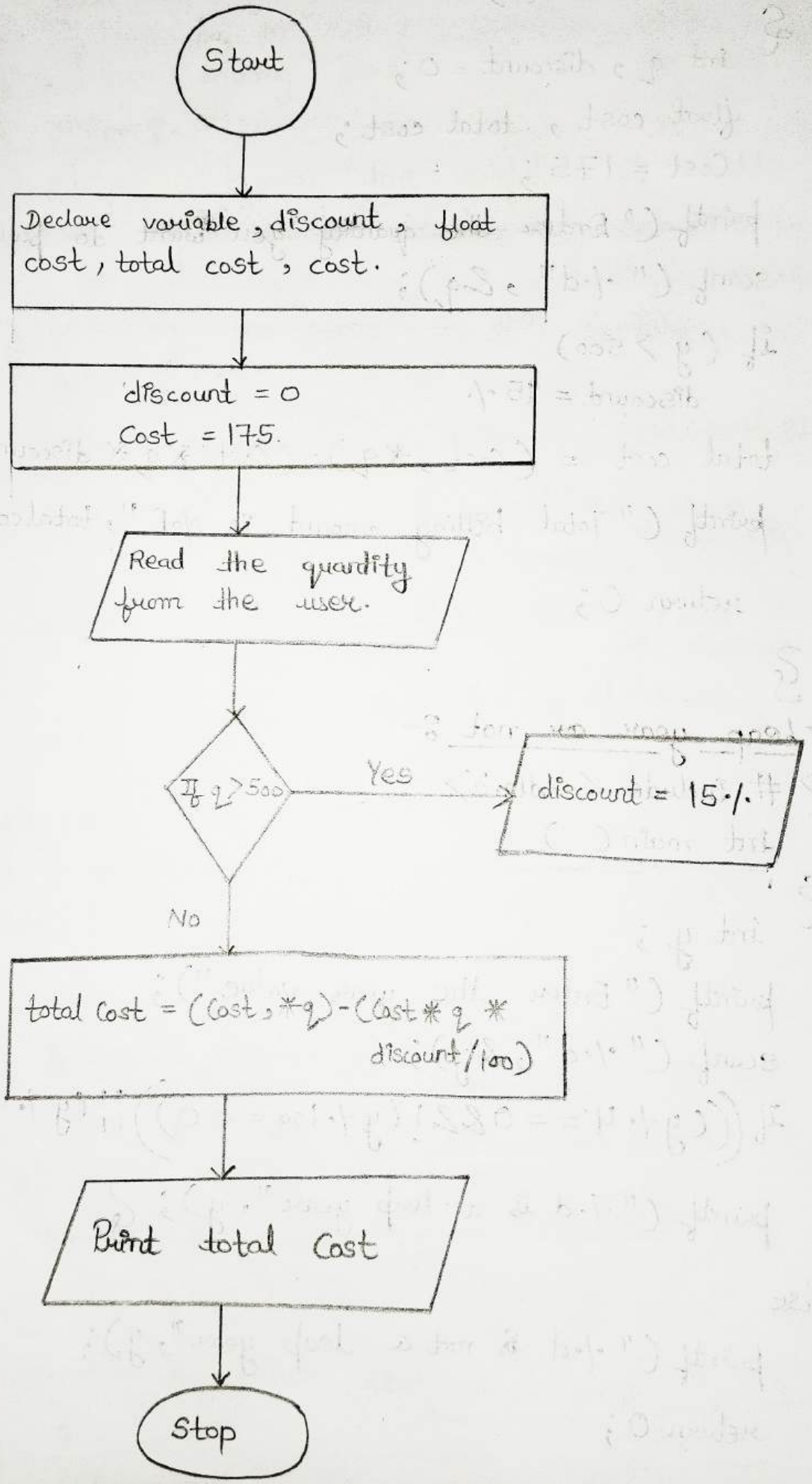
```
=> # include <stdio.h>
   int main( )
{
   int n;
   printf ("Enter a number");
   Scanf ("%d", &n);
   if (n>0)
   printf ("Number is positive");
   else
   printf ("Number is either zero or negative");
}
```

→
```
#include <stdio.h>
int main()
{
    int q, discount = 0;
    float cost, total cost;
    Cost = 175;
    printf("Enter the quantity you want to purchase");
    scanf("%d", &q);
    if (y > 500)
        discount = 15%
    total cost = (Cost, *q) - (Cost *q* discount)/100
    printf("Total billing amount is %of", totalcost);
    return 0;
}
```

★ Leap year or not :-

=>
```
#include <stdio.h>
int main()
{
    int y;
    printf("Enter the year value");
    scanf("%d", &y);
    if(((y%4==0&&!(y%100==0))||(y%400==0))
{ printf("%d is a leap year", y); }
else
    printf("%d is not a leap year", y);
    return 0;
}
```

FLOWCHART...


Start
Declare variable, discount, float cost, total cost, cost.
discount = 0
Cost = 175.
Read the quardity from the user.
If q > 500
Yes
discount = 15%.
No
total cost = (cost * q) - (cost * q * discount/100)
Print total Cost
Stop

★ Leap year or not :-

```
=> # include <stdio.h>
   int main()
   {
     int y;
     printf("Enter the year value");
     scanf("%d", &y);
     if((y%4==0 && !(y%100==0)) || (y%400==0))
   {printf("%d is a leap year", y);}
   else
     printf("%d is not a leap year", y);
     return 0;
   }
```

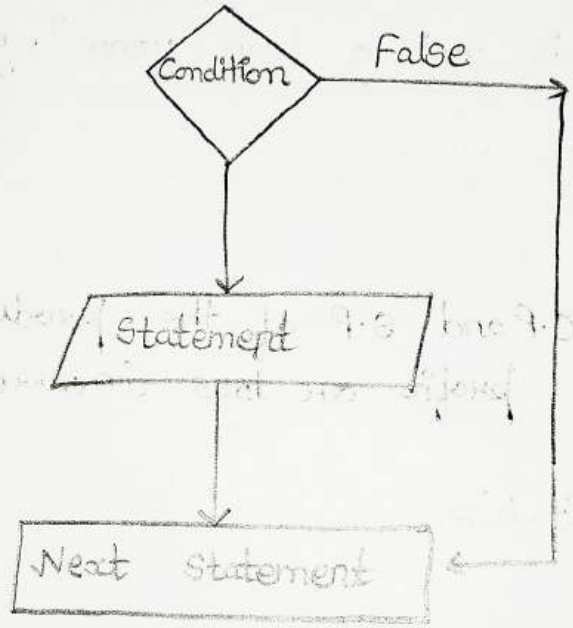
★ WAP to enter C.P and S.P of the product and display the amount of profit or loss incurred by the seller.

```
=> #include <stdio.h>
   Void main()
   {
     float cp, sp, pl;
     printf("Enter cost price of the product");
     scanf("%f", &cp);
     printf("\n Enter selling price of the product");
     scanf("%f", &sp);
     pl = sp - cp;
     if (pl > 0)
   {
     printf("\n Profit = Rs %f", pl);
```

```
}
else
{ if (pl==0)
   printf ("\n No profit no loss");
else
   printf ("\n Loss =Rs %.f", pl * (-1));
}}
```

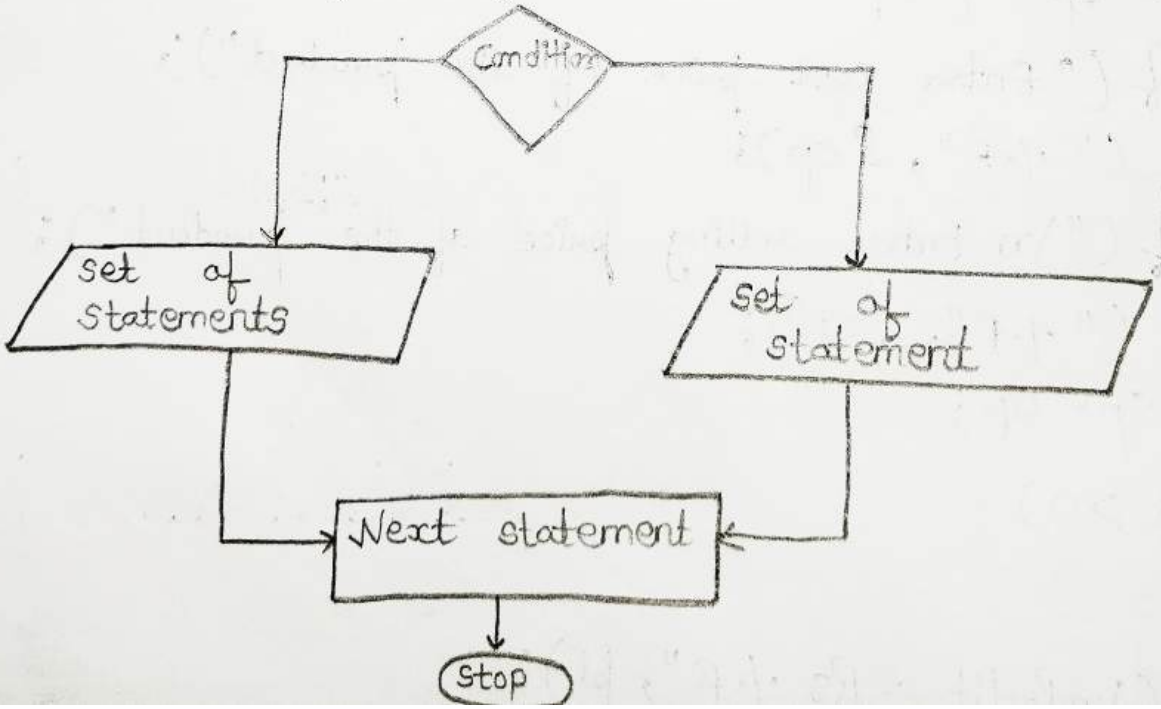
→ Flowchart for only 'if statement' :-

⇒



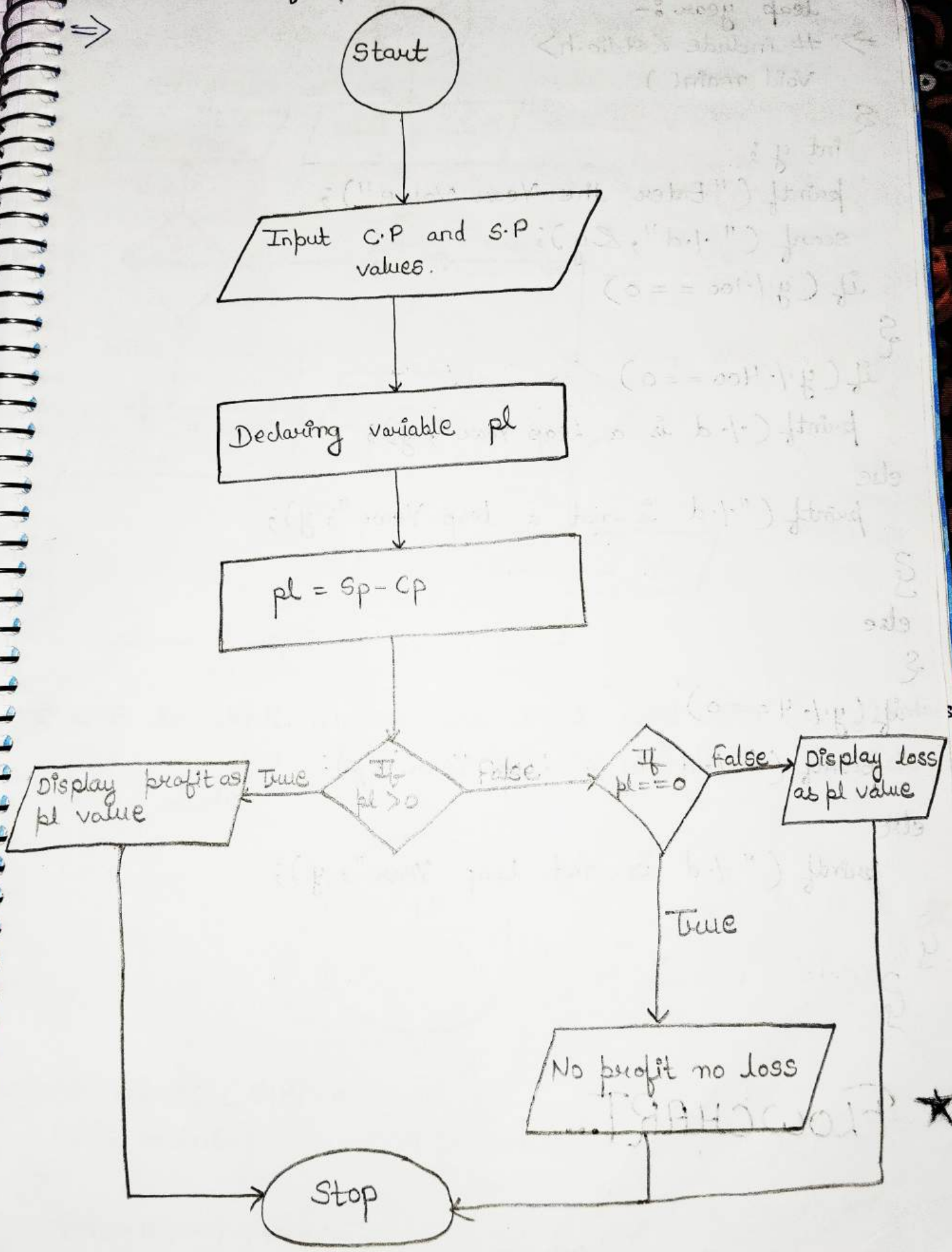
→ Flowchart for if-else statement :-

⇒



★ Flowchart for S.P - C.P statement :-

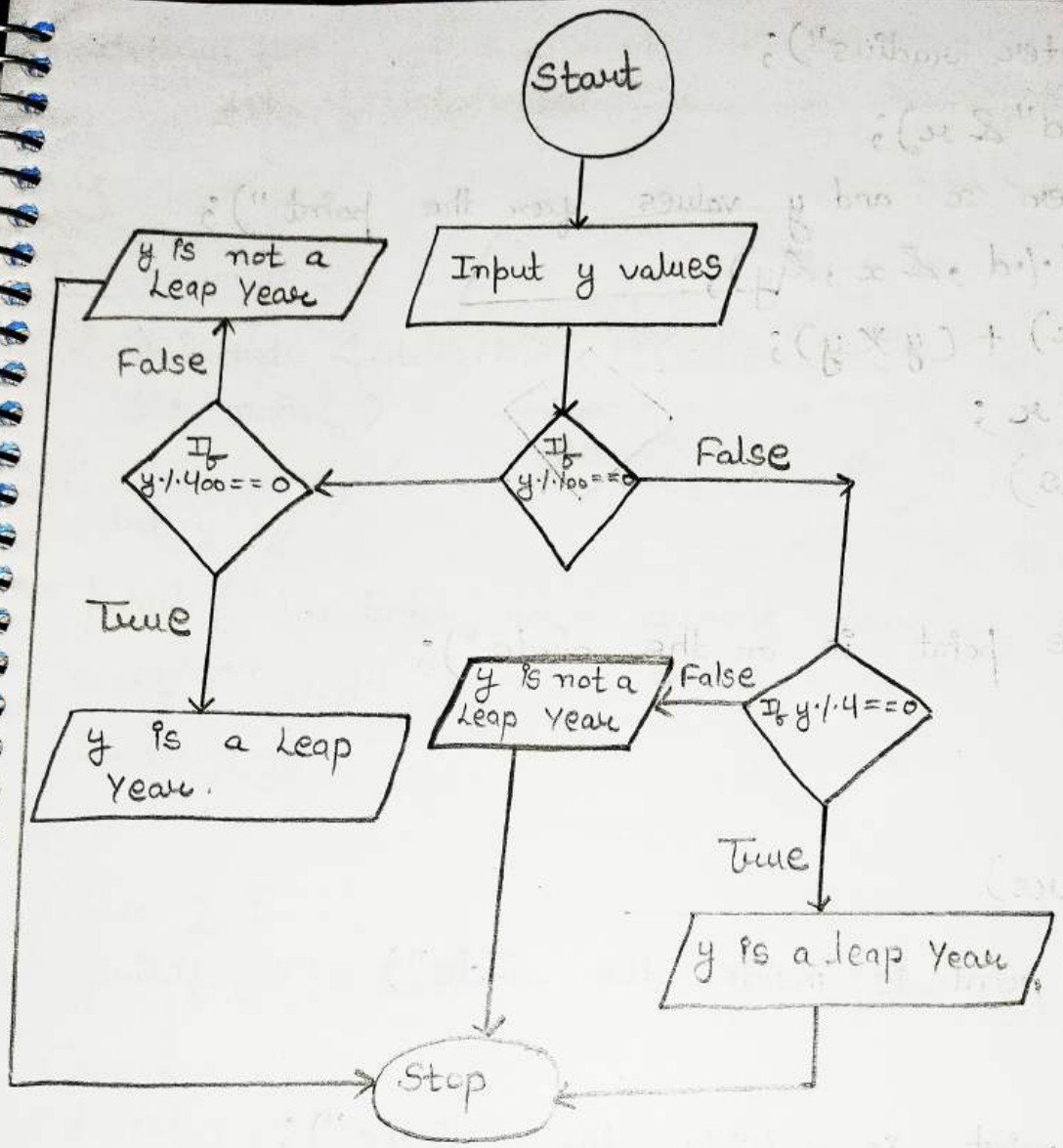
⇒



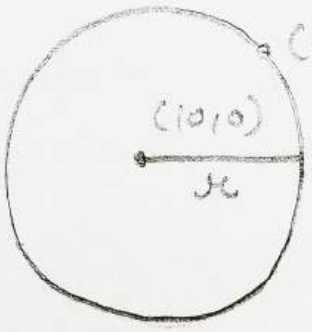
→ WAP to check whether the input Year is leap year :-

```
⇒ #include <stdio.h>
  Void main()
{
   int y;
   printf("Enter the Year Value");
   scanf("%d", &y);
  if (y%100==0)
{
  if(y%400==0)
   printf("%d is a Leap Year", y);
else
   printf("%d is not a leap Year", y);
}
else
{
  if(y%4==0)
   printf("%d is a Leap Year", y);
else
   printf("%d is not Leap Year", y);
}
}
```

★ FLOWCHART...



★ WAP to check whether the input point with (x, y) values lies on the circle, inside the circle or outside the circle.




```
#include <stdio.h>
Void main()
{
int r, x, y, d, rs;
```

```
printf("Enter radius");
scanf("%d" &r);
printf("Enter x and y values for the point");
scanf("%d%d,&x,&y);
d = (x*x) + (y*y);
rs = r*r;
if(d == rs)
{
printf("The point is on the circle");
}
else
{ if (d<rs)
  printf("point is inside the circle")
else
  printf("point is outside the circle");
}
}
```

★ Switch Statement :-

=> Switch (expression)

{

Case Value 1:

set of statements

Case Value 2:

set of statements

```
default:
    set of statement
}
```

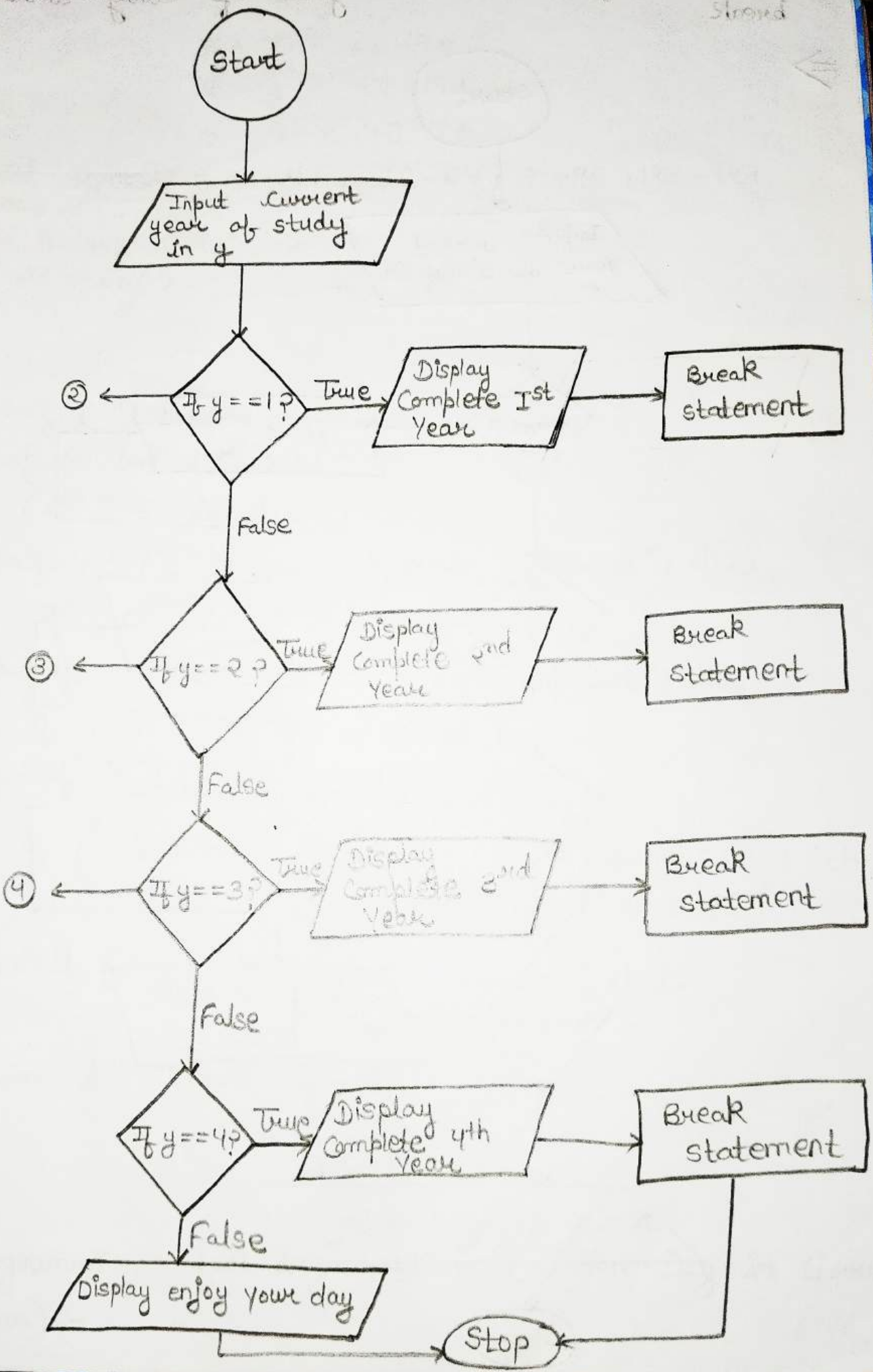
For example :-

```
=> #include <stdio.h>
   Void main()
{
   int y;
   printf("\n Enter your current Year of study");
   scanf("%d", &y);
   Switch (y)
{
   Case 1:
    printf("\n Complete Your first Year of Study");
   Case 2:
    printf("\n Complete Your second Year of study");
   Case 3:
    printf("\n Complete Your third Year of Study");
   Case 4:
    printf(",\n Complete Your Fourth Year of study");
   default:
    printf("\n Enjoy Your Study");
}
}
```

Que. Write a program to check whether the given input is vowel or not using switch case.

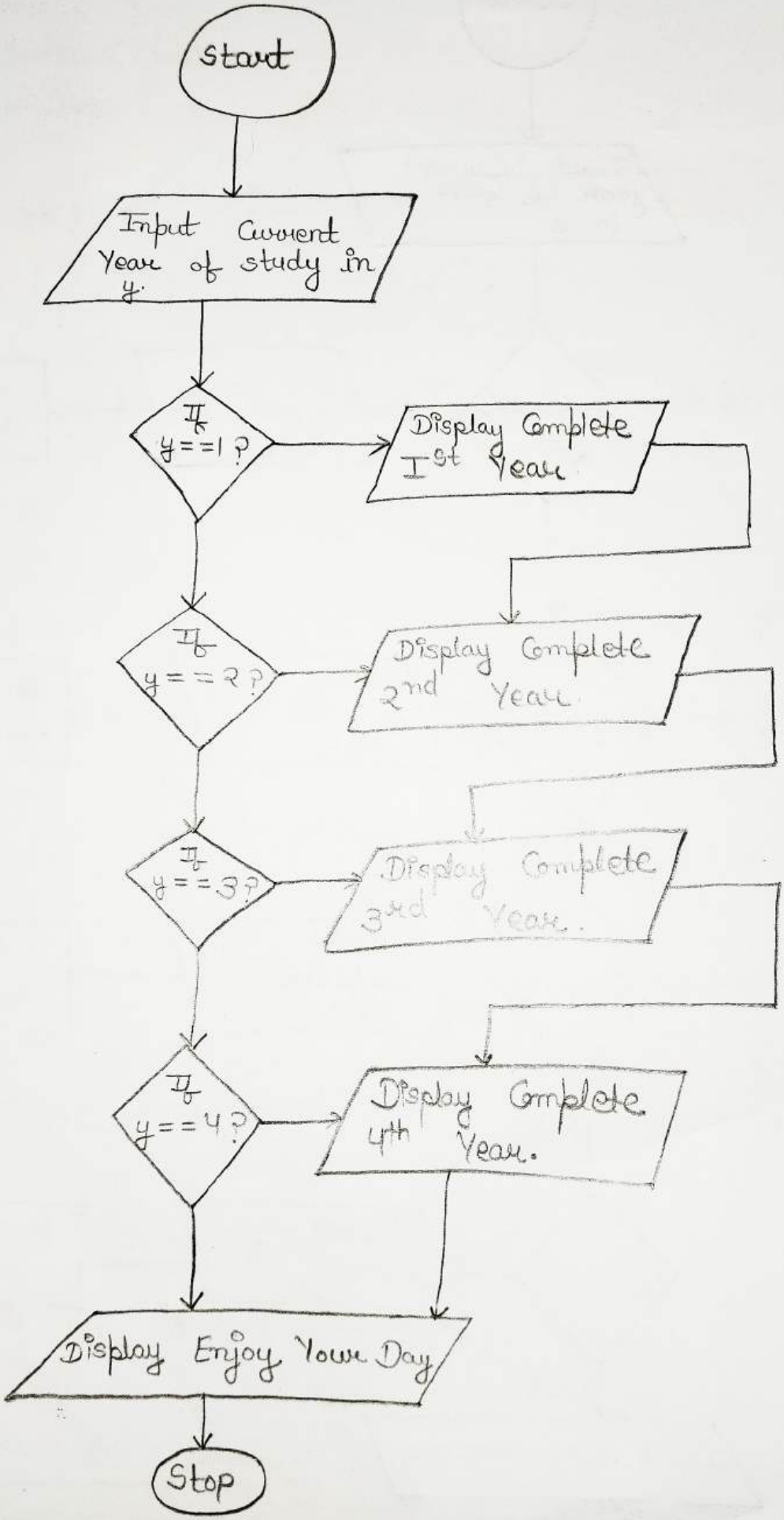
```
=> #include <stdio.h>
   Void main( )
{
  char ch;
  printf (" Enter a character");
  scanf ("%c", &ch);
  Switch (ch)
{
  Case 'A':
  Case 'a':
  Case 'E':
  Case 'e':
  Case 'I':
  Case 'i':
  Case 'O':
  Case 'o':
  Case 'U':
  case 'u':
  printf ("%c is a vowel", ch);
  break;
  default:
  printf ("%c is not a vowel", ch);
}
}
```

→ Flowchart...


Start
Input current year of study in y
If y==1 ?
True
Display Complete Ist Year
Break statement
2
False
If y==2 ?
True
Display Complete 2nd Year
Break statement
3
False
If y==3 ?
True
Display Complete 3rd Year
Break statement
4
False
If y==4?
True
Display Complete 4th Year
Break statement
False
Display enjoy your day
Stop

★ Flowchart for a current year of study without break.

⇒



Skyvalue...

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

Special symbols = 0 - 47, 58 - 64, 91 - 96, 123 - 127

=>
```
# include <stdio.h>
Void main()
{
 char ch;
 printf ("Enter a character");
 Scanf ("%c", &ch);
 if (ch > 64 && ch <91)
 printf ("%c is an uppercase Alphabet", ch);
else if (ch > 96 && ch <123)
   printf ("%c is a lowercase Alphabet", ch);
else if (ch >47 && ch < 58)
   printf ("%c is a digit", ch);
else if ((ch>=0&& ch <48)|| (ch >57&& ch <63 || (ch>
        90&& ch <97)|| (ch > 122&& ch <128))
   printf ("%c is a special symbol", ch);
else
   printf (" Invalid Input");
}
```

★ Programming for day value :- (Julian Day in Current Year) :-

=> #include <stdio.h>

```
Void main()
{
  int d, m, y;
  int julian;
  printf("Enter day, month and Year value for
          date");
  scanf("%d%d%d", &d, &m, &y);
  julian = d;
  switch(m-1)
{
  Case 11: julian += 30;
  Case 10: julian += 31;
  Case 9 : julian += 30;
  Case 8 : julian += 31;
  Case 7 : julian += 31;
  Case 6 : julian += 30;
  Case 5 : julian += 31;
  Case 4 : julian += 30;
  Case 3 : julian += 31;
  Case 2 : julian += 28;
  Case 1 : julian += 31;
}
if((y%4==0 && y%100==0)||(y%400==0))
{ if(m!=1)||(m!=2)
```

7th Jan 2004
d = 7
m = 1
y = 2004

```
    { j++;
    }
}
  printf ("Julian Day for Current Year is %d", j);
```

★ Loop Control Statements / Iteration / Repetition :-

→ for
→ while
→ do-while

for (initialization expression ; Condition ; Update expression)

```
#include <stdio.h>
Void main()
{
  int i;
  for (i=1 ; i <= 20; i++)
  printf ("\n Hello");
}
```

★ WAP to display first n odd numbers :-

=> (1, 3, 5, 7, 9)

```
#include <stdio.h>
Void main()
{
  int i, n;
printf ("Enter no. of odd terms");
scanf ("%d", &n);
for (i=1 ; i <=n; i++)
{ if (i%2!=0)
```

```
    printf ("\n%d", i);
  }
}
```

★ WAP to display Alphabets from A to Z :-

```
=> #include <stdio.h>
   Void main()
{
   char ch;
   for (ch = 'A'; ch <= 90; ch++)
   printf ("%c", ch);
}
```

★ Factorial :-

```
=> #include <stdio.h>
   Void main()
{
   int f = 1, i, n;
   printf (" Input a value from user.");
   scanf ("%d", &n);
   for (i = n; i > 1; i--)
    f = f * i;
   printf (" Factorial of %d is %d", n, f);
}
```

★ WAP to display Fibonacci for series upto N terms :-

=> # include <stdio.h>
Void main()
{
int n;, t1, t2, t;
printf ("Enter number of terms");
scanf ("%d", &n);
t1 = 0;
t2 = 1;
printf ("%d", t2);
for (i = 2; i <= n; i++)
{ t = t1 + t2;
printf ("%d", t);
t1 = t2;
t2 = t;
}

→ Arithmetic operators:-

=> # include <stdio.h>
int main()
{
int a = 67, b = 14, c;
c = a+b;
printf (" a+b = %d \n", c);
c = a-b;
printf ("a-b = %d \n", c);
c = a*b;
printf ("a*b = %d \n", c);
c = a/b;
printf ("a/b = %d \n", c);

```
    c = a%b;
    printf (" Remainder when a divided by b = %d \n", c);
    return 0;
}
```

→ Relational operators :-

```
⇒ # include <stdio.h>
  int main()
{
  int a = 89, b = 85, c = 89;
  printf ("%d == %d is %d \n", a, b, a==b);
  printf ("%d == %d is %d \n", a, c, a==c);
  printf ("%d > %d is %d \n", a, b, a>b);
  printf ("%d > %d is %d \n", a, c, a>c);
  printf ("%d < %d is %d \n", a, b, a<b);
  printf ("%d < %d is %d \n", a, c, a<c);
  printf ("%d != %d is %d \n", a, b, a!=b);
  printf ("%d != %d is %d \n", a, c, a!=c);
  printf ("%d >= %d is %d \n", a, b, a>=b);
  printf ("%d >= %d is %d \n", a, c, a>=c);
  printf ("%d <= %d is %d \n", a, b, a<=b);
  printf ("%d <= %d is %d \n", a, c, a<=c);
  return 0;
}
```

→ Logical operators :-

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

```
int a = 15 , b = -5, c = 23, d;
d = (a==b) && (c>b);
printf ("(a==b) && (c>b) is %d \n", d);
d = (a==b) || (c<b);
printf ("(a==b) || (c<b) is %d \n", d);
d = (a!=b) || (c<d);
printf ("(a!=b) || (c<d) is %d \n", d);
d = ! (a==b);
printf ("! (a==b) is %d \n", d);
d = ! (a!=b);
printf ("! (a!=b) is %d \n", d);
return 0;
}
```

→ Bitwise operators :-

```
=># include <stdio.h>
  int main()
{
  int a = 3 , b = 7;
  printf ("%d & %d is %d \n", a, b , a and b);
  printf ("%d | %d is %d \n", a, b, a |b);
  printf ("%d ^ %d is %d \n", a, b , a ^ b);
  return 0;
}
```

```
=># include <stdio.h>
  int main()
{
  int a = 3 , b=7;
  printf ("%d <<1 is %d\n", a , a <<1);
```

```
printf("%d >> 1 is %d \n", a, a >> 1);
return 0;
}
```

→ Bitwise operator ~ stands for 2's complement :-

```
#include <stdio.h>
int main()
{
  int a = 5;
  printf("~%d is %d \n", a, ~a);
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  unsigned int a = 5;
  printf("~%u is %u \n", a, ~a);
  return 0;
}
```

→ Assigment operators :-

```
#include <stdio.h>
int main()
{
  int a = -76;
  int c;
  c = a;
  printf("c = %d\n", c);
  c += a;
  printf("c = %d\n", c);
  c -= a;
  printf("c = %d\n", c);
  c *= a;
```

```
printf ("c = %d\n", c);
c /= a;
printf ("c = %d\n", c);
c %= a;
printf ("c = %d\n", c);
return 0;
}
```

→ Unary operator :-

```
#include <stdio.h>
int main()
{
  int a = 3, b = 7;
  float c = 4.5, d = 8.5;
  printf ("++a = %d \n", ++a);
  printf (" b-- = %d \n", b--);
  printf ("c++ = %f \n", c++);
  printf ("--d = %f \n", --d);
  return 0;
}
```

→ Ternary operator :-

```
#include <stdio.h>
int main()
{
  int a, b, c;
  printf (" Enter two numbers ");
  scanf ("%d%d", &a, &b);
  c = (b == 0) ? 0 : (a/b);
  printf ("c = %d\n", c);
  return 0;
}
```

=> #include <stdio.h>
int main()
{
int a, b, c;
printf("Enter two numbers");
scanf("%d%d", &a, &b);
c = (b==0)?0 : (a/b);
printf("c = %d\n", c);
return 0;
}

→ WAP to find series upto 1+2+4+7+11+16+ ---- upto N terms.

i	t = 1 (i++)	sum = 0 sum=(sum+t)	Code

0	0+1 = 1	1	#include <stdio.h>
			Void main()
			{ int i, t=1, sum=0, n;
			printf("Enter the number of terms");
			scanf("%d", &n);
			for (i=0; i<n; i++)
			{
			t+ = i;
			sumt = t;
			}
			printf("Sum of %d terms is %d", n, sum);
			}

1	1+1 = 2	1+2	
2	2+2 = 4	1+2+4	
3	3+4 = 7	1+2+4+7	
4	4+7 = 11	1+2+4+7+11	
5	5+11 = 16	1+2+4+7+11+16	

★ While loop :-

=> while (condition)
{
  set of statements
}

→ WAP to enter numbers and display their cubes until you enter number zero.

```
=> #include <stdio.h>
   Void main()
{
   int n;
   printf ("Enter a number");
   scanf ("%d", &n);
  while (n!=0)
{ printf ("Cube of %d is %d", n, n*n*n);
   printf ("Enter the number");
   scanf ("%d", &n);
}
}
```

→ do-while loop :-

```
=> do
   { set of statement
   }
  while (condition);
```

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

```
int n;
do
{
printf ("Enter the number");
scanf ("%d", &n);
if (n == 0)
break;
else
printf ("Cube of %d is %d", n, n * n * n);
}
while (n != 0);
}
```

→ WAP to find a number raised to power another number using while:-

```
=># include <stdio.h>
Void main()
{
long int result;
int b, p, i;
result = 1;
i = 1;
printf ("Enter the number");
scanf ("%d", &b);
printf ("Enter the power with which number to
be raised");
scanf ("%d", &p);
while (i<=p)
```

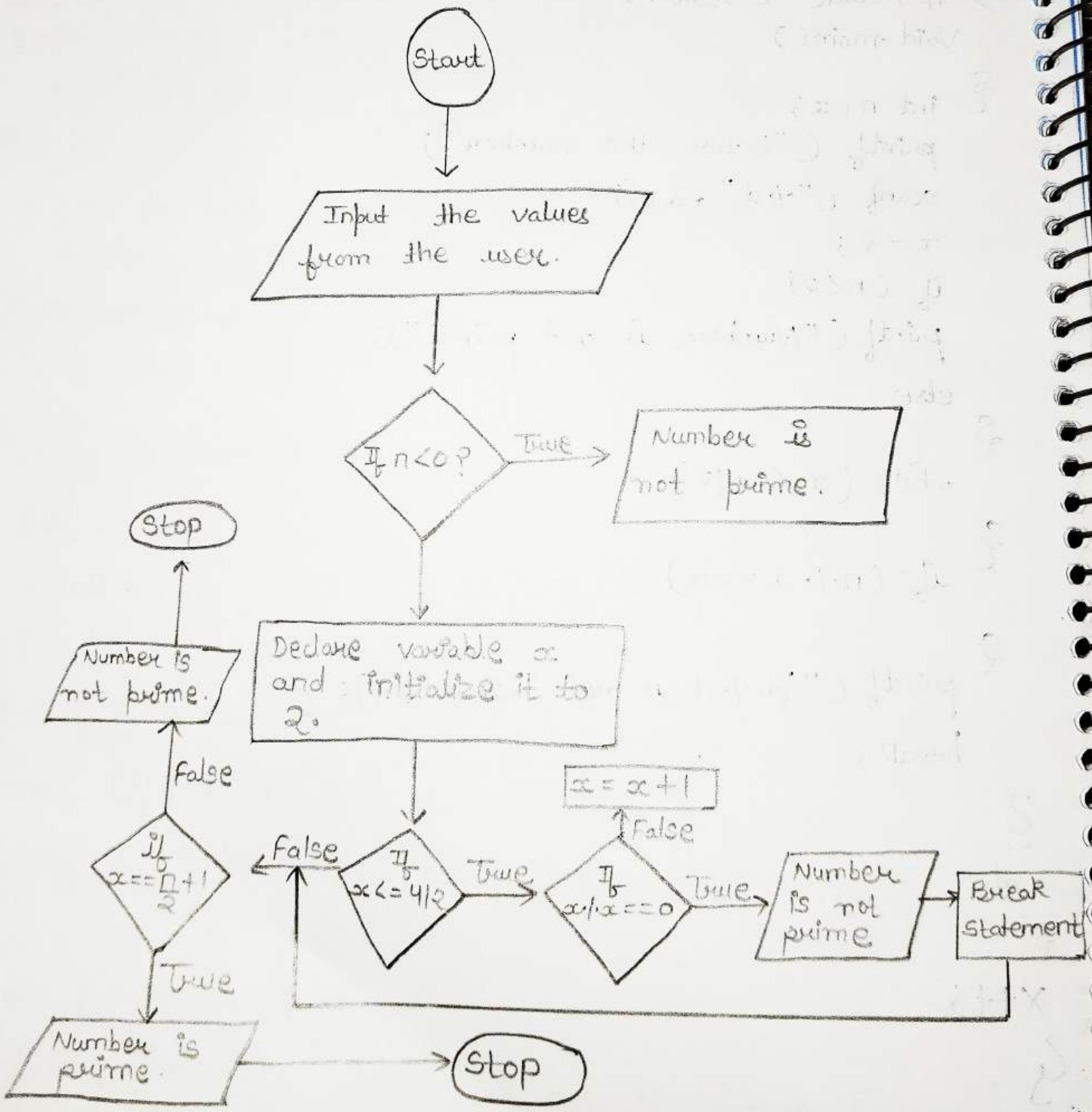
```
{
    result = result * b;
}
printf ("Number %d raised to power %d is %ld", b,
        p, result);
}
```

→ WAP to check input number is prime or not using while loop.

```
⇒ # include <stdio.h>
  Void main()
{ int n, x;
  printf ("Enter the number");
  scanf ("%d", &n);
  x = 2;
  if (n<2)
  printf ("Number is not prime");
  else
  {
  while (x <= n/2)
  {
  if (n % x == 0)
  {
  printf ("\n%d is not Prime", n);
  break;
  }
  else
  {
  x++;
  }
}
```

```
if (x == n/2+1)
{
  printf("%d is prime", n);
}
}
}
```

→ FLOWCHART...



→ WAP to Calculate number of digits in a number using do while loop.

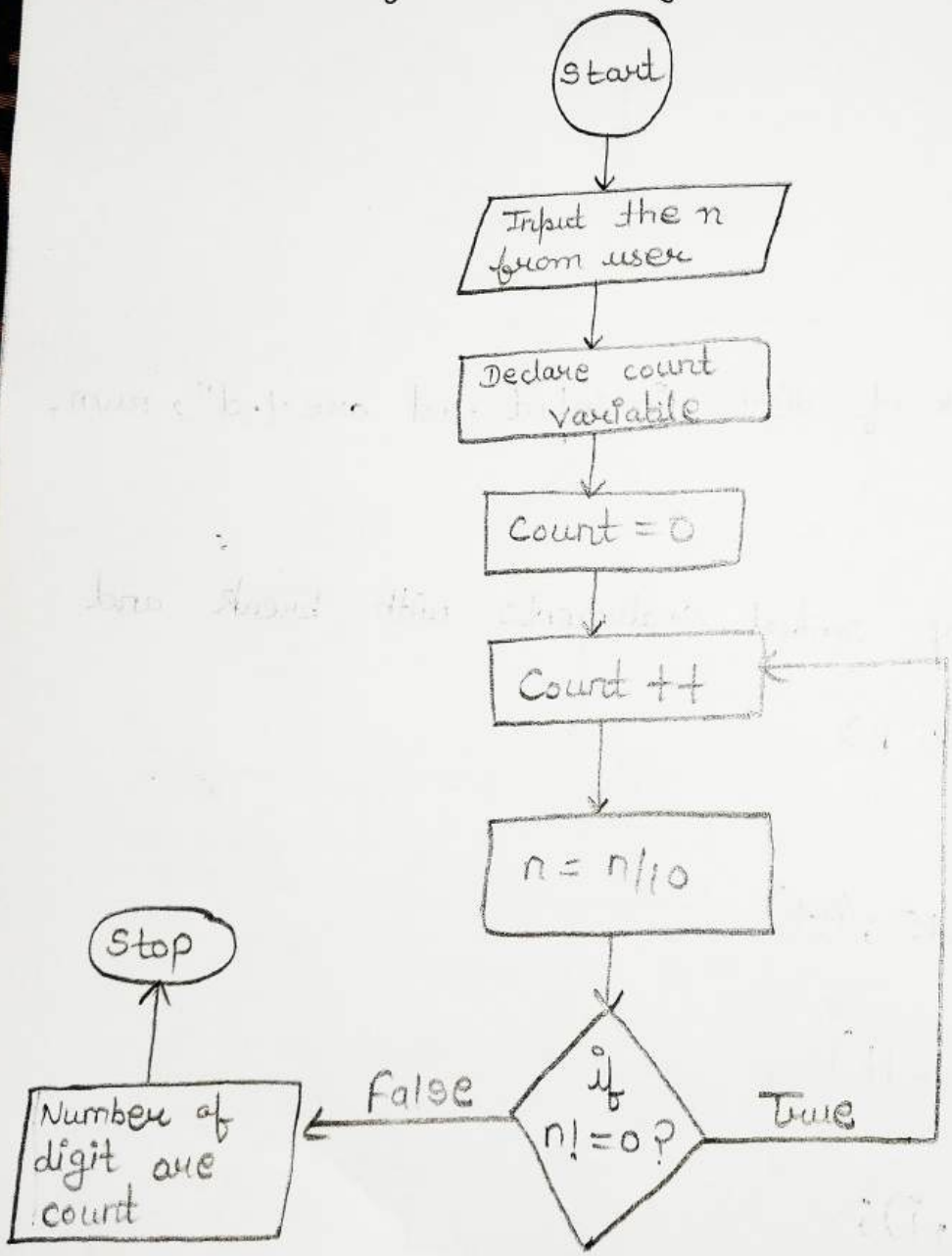
```
=> # include <stdio.h>
   # include <math.h>
   Void main( )
{
   int n, num, count=0;
   printf ("Enter the number");
   scanf ("%d", &n);
   num = n;
   do
{
   count ++ ;
   n = n/10;
}
   while (n!=0);
   printf ("\n number of digits in input %d are %d", num,
          count);
}
```

→ Examples of loop control statements with break and continue.

```
=> # include <stdio.h>
   Void main ()
{
   for (int i=0; i<15; i++)
{
   if (i==7 || i==11)
   break;
   printf ("%d ", i);
}
```

```
printf ("\n");
for (int j=0; j<15; j++)
{
    if (j==7 || j==11)
    Continue;
    printf ("%d", j);
}
}
```

→ Flowchart for Count digit.



```
# include <stdio.h>
Void main()
{
  int n , sum=0 , i = 0;
  printf ("enter the number of terms");
  scanf ("%d", &n);
  labelsum : sum = sum+ i;
  if (i<n)
  {
    i++;
    goto labelsum;
  }
  printf ("sum of first %d natural number is %d", n)
}
```

```
=># include <stdio.h>
  Void main()
{
  int n , r , num , i= 1;
  printf ("Enter number");
  scanf ("%d", &n);
  num = n;
  while (n!=0)
{
  r = n%2;
  bin = bin+ r* i;
  n = n/2;
  i = i * 10;
}
  Printf ("Binary equivalent of %d is %ld", num, bin);
}
```

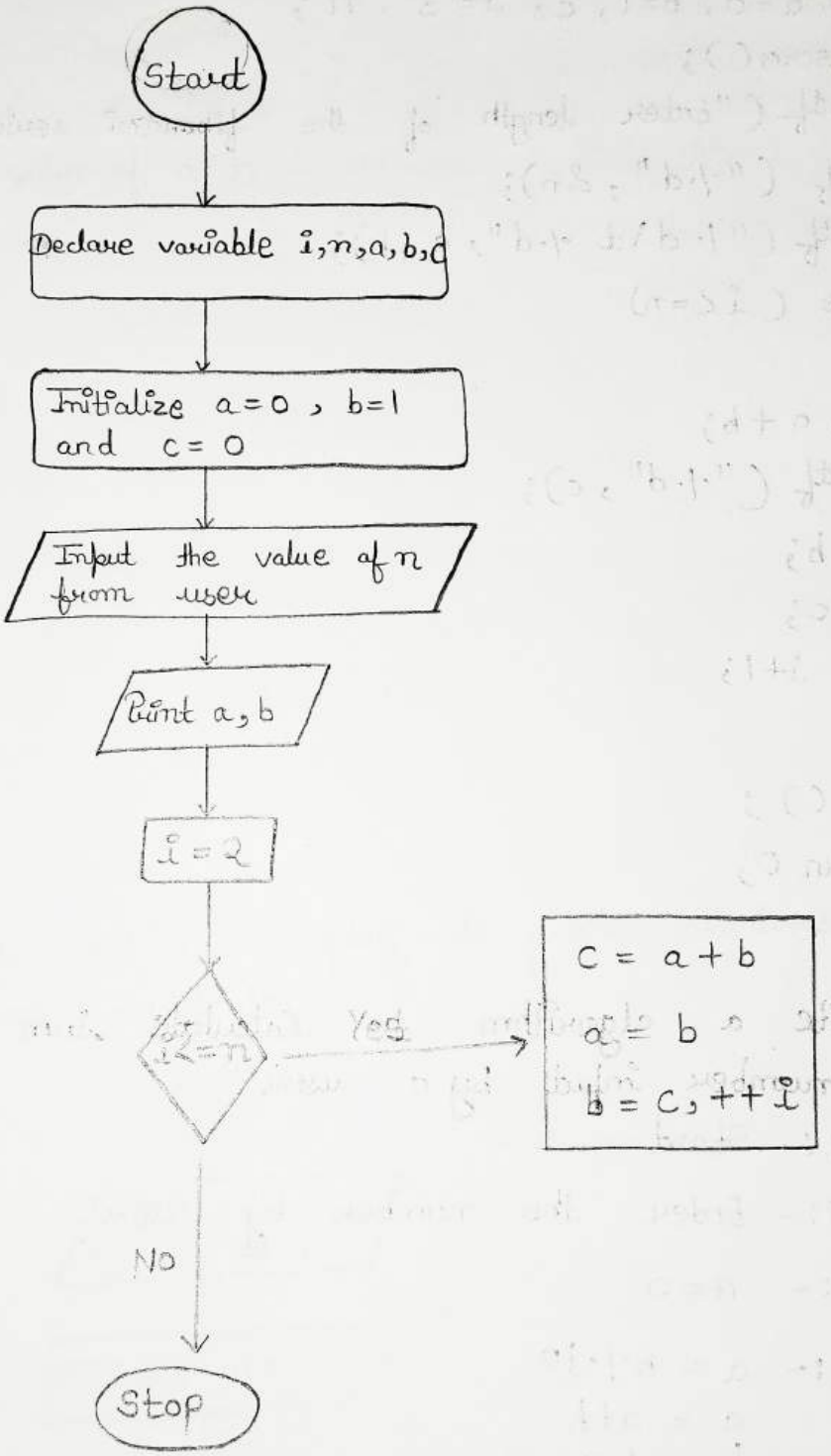
```
#include <stdio.h>
void main()
{
long int rev = 0;
int n, num, r;
printf ("enter the number");
scanf ("%d", &n);
num = n;
while (n!=0)
{
r = n%10;
rev = rev * 10 + r;
n = n/10;
}
printf ("reverse of %d is %ld", num, rev);
if (num == rev)
printf ("\n%d is palindrome", num);
else
printf ("\n%d is not palindrome", num);
}
```

⇒ Difference b/w for, while, do while :-

⇒

Assignment =>2

1. Design a flowchart to display Fibonacci series upto n (a) number of terms.



(b) WAP in C to implement above problem using while loop.

=>
```
# include <stdio.h>
# include <conio.h>
int main()
{
int a=0, b=1, c, i= 3 , n ;
clrscr();
printf ("Enter length of the fibonacci series");
scanf ("%d", &n);
printf ("%d \t %d", a , b);
while ( i<=n)
{
c = a+b;
printf ("%d", c);
a = b;
b = c;
i = i+1;
}
getch () ;
return 0;
}
```

2.(a) Write a algorithm to calculate sum of digits of a number input by a user

=> Step 1:- Start

Step 2:- Enter the number by user

Step 3:- a=0

Step 4:- a = b%10
a = a+b
b = b/10

Step 5:- if (b>0)

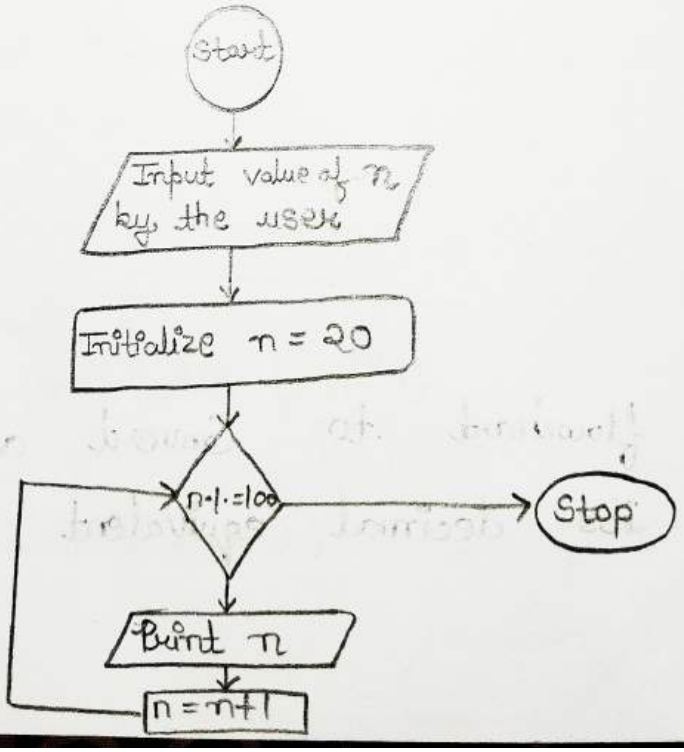
Step 6:- Display a

Step 7:- Stop

(b) WAP in C to implement above problem using while loop.

```
#include <stdio.h>
#include <conio.h>
int main()
{
int a, b;
clrscr();
printf("Enter value of a");
scanf("%d", &a);
b = 0;
do
{
b += a%10;
a /= 10
}
while (a>0);
printf("The sum is = %d \n", b);
getch();
return 0;
}
```

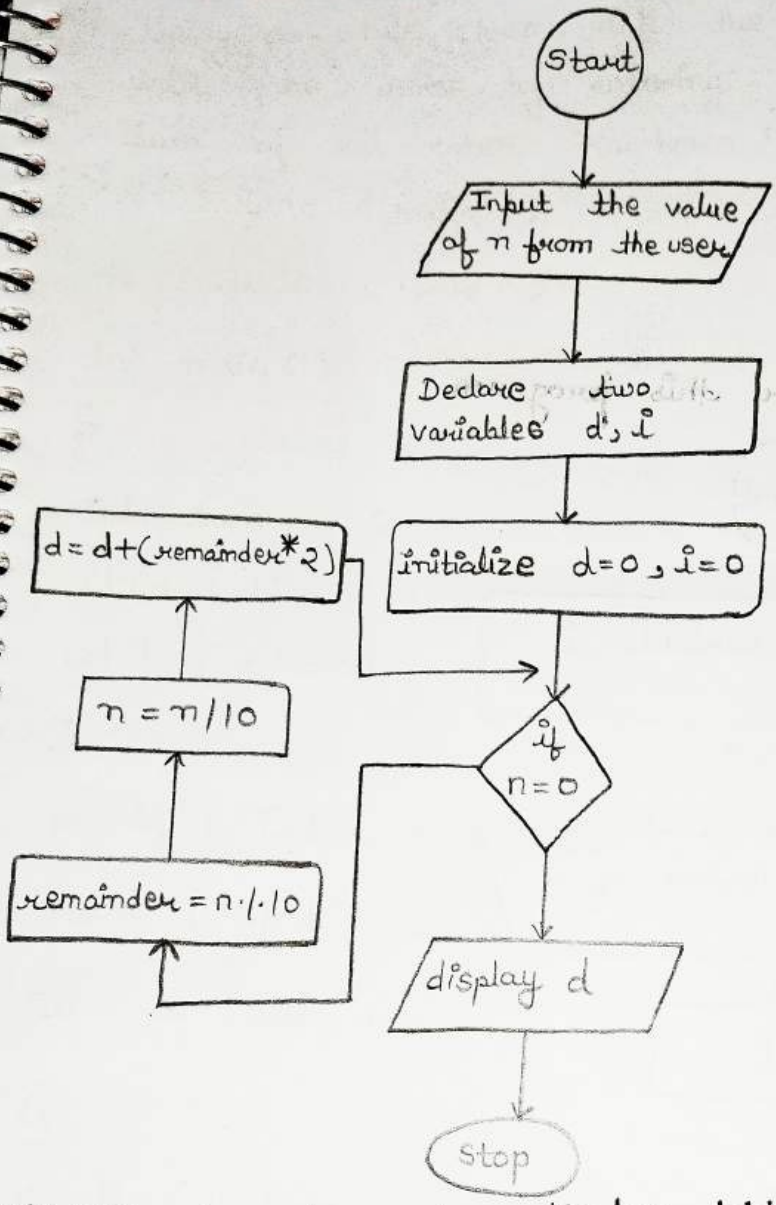
3. (a) Design a flowchart to display all prime numbers starting from a value 20 to 100



(b) WAP in C to implement the above problem

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int n, i, show;
clrscr();
printf("All prime numbers from 20 to 100");
for (n=2 ; n<=100 ; n+1)
{
 show = 0;
for ( i=1 ; 1<=n ; i++)
{
 if (n%. i==0)
{
 show ++;
}
}
if (Count==2)
{
printf("%5d",n);
}
}
getch();
}
```

5. Design a flowchart to Convert a binary number to its decimal equivalent

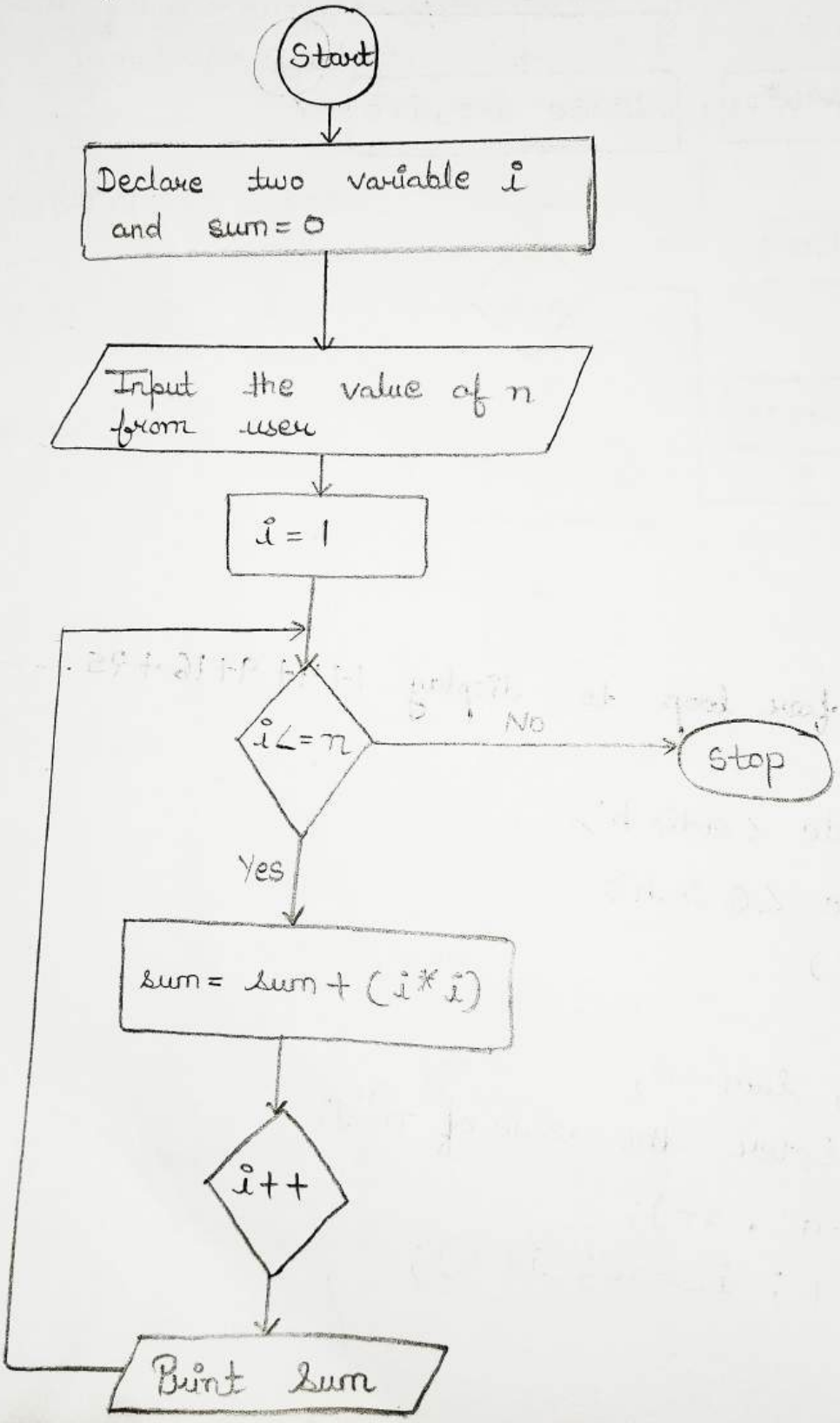


6 (a) Use for loop to display 1+4+9+16+25 ---- upto n terms.

=>
```
# include <stdio.h>
# include <Conio.h>
int main()
{
int n, i, sum=0;
printf ("Enter the value of n");
scanf ("%d", &n);
for (i=1; i<=n; i++);
{
```

```
    sum = sum + (i * i);
}
printf ("sum = %d", sum);
getch();
return 0;
}
```

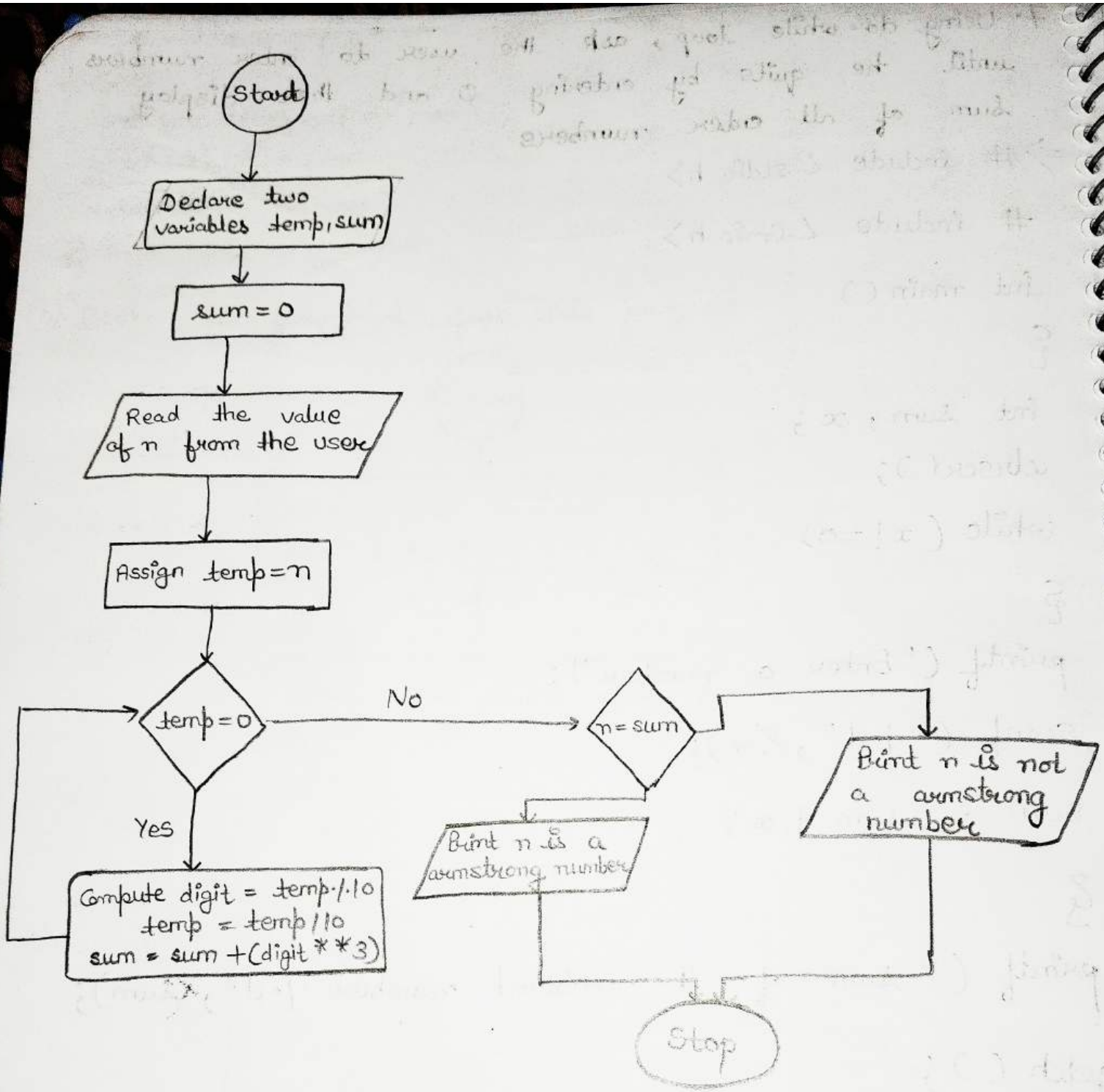
(b) Design the flowchart for this program



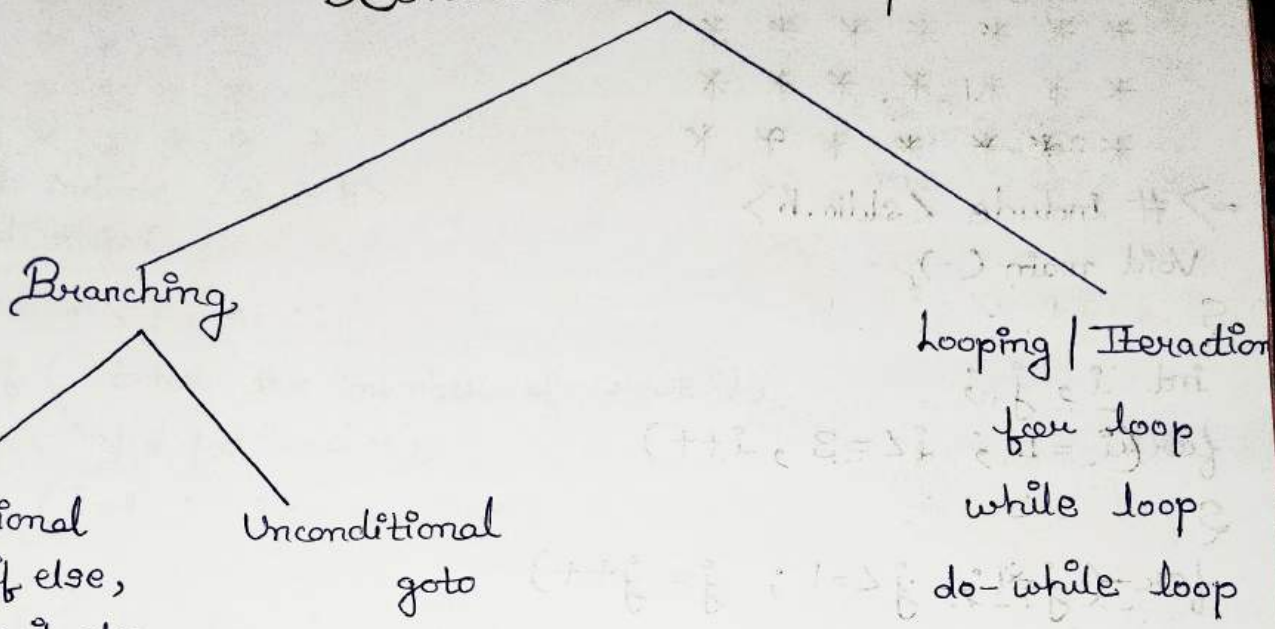
7. Using do-while loop, ask the user to enter numbers until he quits by entering 0 and then display sum of all enter numbers

```
#include <stdio.h>
#include <conio.h>
int main()
{
int sum, x;
clrscr();
while (x!=0)
{
printf ("Enter a number");
scanf ("%d", &x);
sum = sum + x;
}
printf ("sum of the entered number %d", sum);
getch();
return 0;
}
```

8. Draw a flowchart to check whether given number is Armstrong number or not


Start
Declare two variables temp, sum
sum = 0
Read the value of n from the user
Assign temp = n
temp = 0
No
Yes
Compute digit = temp % 10
temp = temp // 10
sum = sum + (digit ** 3)
n = sum
Print n is a armstrong number
Print n is not a armstrong number
Stop

Control Statement / Statement



→ 1 3 5 7 9 11

⇒
```
# include <stdio.h>
Void main()
{
 int i, j;
 for (i=1; i<=5; i++)
{
 for (j=1, j<=11; j=j+2)
{
 printf ("%d", &j);
}
printf ("\n");
}
}
```

→ Display :-

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

```
=> # include <stdio.h>
  Void main ( )
{
 int i, j ;
 for(i =1 ; i<=3 , i++)
 {
 for (j=1; j<=1 ; j= j++)
 {
  printf ("*");
 }
 printf ("\n");
 }
 }
```

→ Rows and columns :-

```
=> # include <stdio.h>
  Void main()
{
 int , i, j , r, c;
 printf ("Enter the number of rows and columns");
 scanf ("%d%d", &r, &c);
for (i =1 ; i <=r ; i++)
{
for ( j=1 ; j <=c ; j= j++)
{printf ("*");
}
 printf ("\n");
}
```

→
```
*
* *
* * *
* * * *
* * * * *
```

```
#include <stdio.h>
Void main( )
{ int i, j, n, c;
printf ("Enter the number of rows");
scanf ("%d%d", &n);
for (i=1 ; i<=n ; i++)
{
for (j=1; j<=i ; j++)
{
printf ("*");
}
printf ("\n");
}
}
```

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

```
#include <stdio.h>
Void main()
{int i, j, n, c;
printf (" Enter the number of rows");
scanf ("%d%d", &n);
for (i=1 ; i<=n ; i++)
{
for (j=1 , j<=n-i+1 ; j++)
```

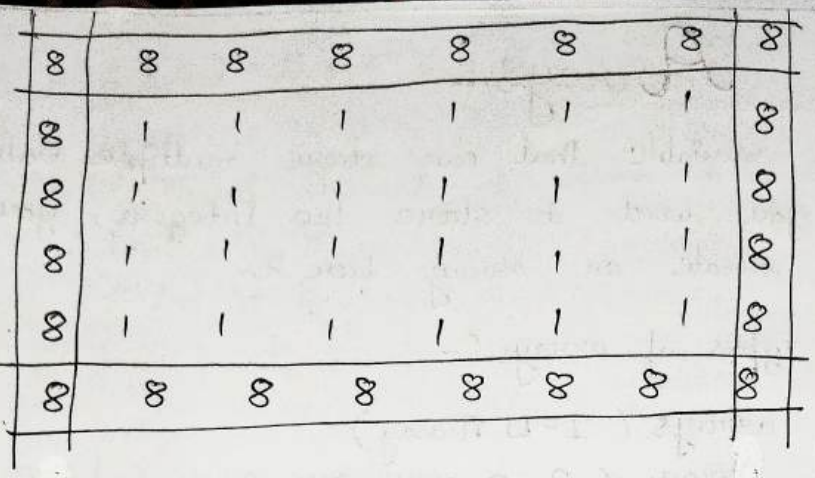
```
{printf ("%d ", j);
}
printf ("\n");
}
}
```

→ Elyod's triangle :-

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

```
=> #include <stdio.h>
  Void main()
  {
  int i, j, n, c = 1;
  printf (" Enter number of rows");
  scanf ("%d%d", &n);
  for (i=1; i<=n; i++)
  {
  for (j<=1; j+=1; c++; j=1)
  {
  printf ("%d", c);
  }
  printf ("\n");
  }
  }
```

→ padding :-

→ 

```
# include <stdio.h>
Void main()
{
int i, j, r, c;
char b, p;
printf ("Enter the character for making border");
scanf ("%c", &b);
printf ("Enter the character for padding")
scanf ("%c", &p);
printf ("Enter the number of rows and columns");
scanf ("%d%d", &r, &c);
for (i = 1; i <= r; i++)
{
for ( j=1; j<=c ; j++)
{
if ( i== 1 || i == r || j == 1 || j == c)
printf ("%c", b);
else
printf ("%c" , p);
}
printf ("\n");
}
}
```

Arrays...

=> An array is a variable that can store multiple values.

For example :- If you want to store 100 integers, you can create an array for it.

→ There are 3 types of arrays :-

1. One dimensional arrays (1-D Array)
2. Two dimensional arrays (2-D Array)
3. Three dimensional arrays (3-D Array)

1: One-dimensional arrays :-

=> The 1-D arrays have only one-dimension. They are represented by a single row or column.

→ Syntax of 1-D Array => datatype arrayname [size];

2. Two-dimensional arrays :-

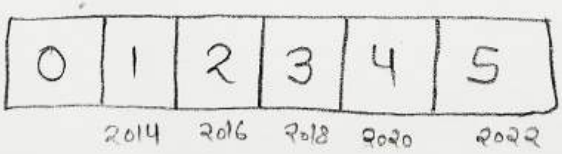
=> The 2-D arrays can be visualised in rows and columns organized in a two-dimensional plane.

→ Syntax of 2-D Array => datatype arrayname [size 1] [size 2];
datatype arrayname [rows] [columns];

3. Three-dimensional arrays :-

=> 3-D arrays have three dimensions and can be visualised as a collection of 2-D arrays organized one on the other to create a third dimension.

→ Syntax of 3-D Array => datatype arrayname [size 1] [size 2] [size 3];



→ WAP for odd numbers

```
=> # include <stdio.h>
   Void main()
```

```
{
int odd [5] = {1,3,5,7,9};
int i;
printf ("Third element of array is %d", odd [2];
for (i=0; i<5 ; i++)
printf ("%d \t %d\n", i, odd [i]);
odd [4] = 17; // 1,3,5,7,17
}
```

→ WAP for even numbers

```
=># include <stdio.h>
Void main()
{
int even [5];
for (i=0; i<5; i++)
even [i] = odd [i] +1; // 2,4,6,8,18
printf ("%d \t %d \n", i, even [i]);
}
```

→ WAP to input array elements for user, display sum of all the elements, maximum value and minimum value in array elements

min max

{-10, 3, 11, 21, -50, 100, 4, 33, 12, 17}

```
=># include <stdio.h>
Void main()
{
int sum [10], total=0, max, min, i;
printf ("Enter array elements");
for (i=0; i<10; i++)
{ scanf ("%d", & sum [i]);
total = total + sum [i]; }
printf ("sum of array elements is %d", total);
min = sum [0];
max = sum [0];
```

```
for (i=1; i<=9; i++)
{
 if (sum [i] < min)
    min = sum [i];
 if (sum [i] > max)
    max = sum [i];
}
printf ("maximum value is %d", max);
printf ("minimum value is %d", min);
}
```

→ WAP to search an element in the given array {-10, 3, 21, -50, 100, 4, 33, 12, 17}

```
=> # include <stdio.h>
   Void main ()
 {
  int array1 [10], x, i;
  printf ("Enter array elements");
  for (i=0; i<10; i++)
  scanf ("%d", &array1 [i]);
  printf ("Enter the element to be searched");
  scanf ("%d", &x); 114
  for (i=0; i<10; i++)
 { if (x == array1 [i])
 { printf ("%d is found at location %d", x, i);
  break;
 }
 }
```

Found

If not found,

```
if (i==10 && array1[9]!=x)
printf ("%d not found", x);
}
```

→ WAP to delete an element from the given array

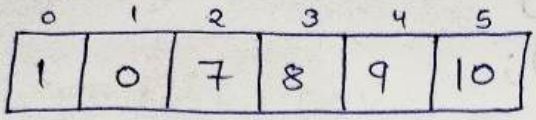
⇒ { -10, 3, 21, -50, 00, 17, 33, 12, 13, 4 }
0 1 2 3 4 5 6 7 8 9

⇒ -10, 3, 21, -50 | 17, 33, 12, 13, 14
0 1 2 3 | 4 5 6 7 8

```
#include <stdio.h>
Void main()
{
int array1[10], x, i, j, count=0;
printf ("Enter array elements");
for (i=0; i<10; i++)
scanf ("%d", &array1[i]);
printf ("Enter the element to be searched");
scanf ("%d", &x); // 4
for (i=0; i<10; i++)
{ if (x== array1[i])
{ printf ("%d is found at location %d", x, i);
break;
}
}
if (count==1)
for (j=i; j<10; j++)
{
array1[j] = array[j+1];
}
if (count==0)
printf ("%d not found", x);
}
```

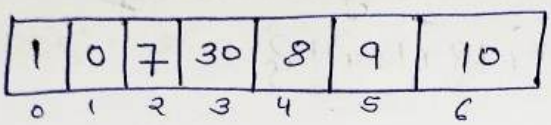
★ WAP to insert an element in the array

⇒



x = 30

loc = 3




```
# include <stdio.h>
 Void main()
{
 int a[30], x, loc, i, n;
 printf ("Enter number of elements for the array");
 scanf ("%d", &n);
 printf ("Enter array elements");
 for (i=0; i<n; i++)
 scanf ("%d", &a[i]);
 printf ("Enter the element to be inserted");
 scanf ("%d", &x);
 printf ("Enter the index value to place the element");
 scanf ("%d", &loc);
 for (i = loc+1 ; i<n+1 ; i++)
{ a[i] = a[i-1]; }
  a[loc] = x;
printf ("Array element after insertion");
for (i=0 ; i<n+1 ; i++)
printf ("%d", a[i]);
}
```

★ 2D Arrays :—

Example :-

rows \ columns	0	1	2	3
0	3	2	1	5
1	7	11	-10	15
2	30	100	55	71

```
# include <stdio.h>
Void main()
{
int x[3][4], i, j;
printf ("Entere array elements");
for (i=0 ; i<=2 ; i++)
{
for (j= 0; j<= 3 ;j++)
scanf ("%d", &x[i][j];
}
for(i=0 ; i<=2 ; i++)
{
for (j=0 ; j<=3; j++)
{
printf ("%d", x[i][j]);
}
printf ("\n");
}
}
```

→ Odd n of matrix :-

```
=># include <stdio.h>
Void main()
{
int x[10][10], y[10][10], z[10][10];
int i, j, m, n;
```

```
printf ("Enter elements for Matrix 1");
for (i=0 ; i<m ; i++)
{
 for (j=0 ; j<n ; j++)
 scanf ("%d", &x [i][j]);
}
printf ("Enter elements for Matrix 2");
for (i=0 ; i<m ; i++)
{
for (j=0 ; j<n ; j++)
scanf ("%d", & y [i][j]);
}
for (i=0 ; i<m ; i++)
{
 for (j=0 ; j<n ; j++)
{
 z [i][j] = x [i][j] + y [i][j];
 printf ("%d", z [i][j]);
}
 printf ("\n");
}
}
```

→ Transpose of matrix :-

3	7	30
2	11	100
1	-10	55
5	15	71

```
#include <stdio.h>
Void main()
{
int x[10][10], Tx[10][10];
int i, j, m, n;
printf("Enter Matrix order as mxn");
scanf("%d%d", &m, &n);
printf("Enter elements for Matrix");
for(i=0; i<m; i++)
{
for(j=0; j<n; j++)
scanf("%d", &x[i][j]);
}
for(i=0; i<n; i++)
{
for(j=0; j<m; j++)
Tx[i][j] = x[j][i];
}
printf("Transpose of Matrix is");
for(i=0; i<n; i++)
{
for(j=0; j<m; j++)
printf("%d", Tx[i][j]);
printf("\n");
}
}
```

→ Multiplication of Matrix :-

$$\begin{matrix} & \begin{matrix} 0 & 1 \end{matrix} \\ \begin{matrix} 0 \\ 1 \end{matrix} & \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \end{matrix} \quad \begin{matrix} & \begin{matrix} 0 & 1 & 2 \end{matrix} \\ \begin{matrix} 0 \\ 1 \end{matrix} & \begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 1 \end{bmatrix} \end{matrix}$$

$$\begin{bmatrix} 2\times1+3\times1 & 2\times2+3\times3 & 2\times4+3\times1 \\ 4\times1+5\times1 & 4\times2+5\times3 & 4\times4+5\times1 \end{bmatrix}$$

```
#include <stdio.h>
Void main()
{
int i,j, sum=0;
printf ("Enter elements for Matrix 1");
for (i=0; i<2; i++)
{
for (j=0; j<2; j++)
scanf ("%d", &x[i][j]);
}
printf ("Enter elements of Matrix 2");
for (i=0; i<2; i++)
{
for (j=0; j<3; j++)
scanf ("%d", &y[i][j]);
}
```

Assignment ⇒ 3

Q1. objective type / short – answer type questions :-

(i) which of the following are invalid variable names and why ?

a) xdigit

b) –num

c) register

d) Count

e) 5_digrum

f) Pos num

g) HEIGHT

h) X

⇒ Invalid variables are :- 5_digrum, register and Pos num

Because in 5_digrum this variable contain both character and number which is not a valid variable name, Register is a storage class and Pos num variable contain space between them which is also invalid.

(ii) Write a program to check given number is even or odd

```
⇒# include <stdio.h>
# include <Conio.h>
Void main()
int n;
clrscr();
printf ("Enter a number");
scanf ("%d", &n);
if (n%2==0)
{
printf ("%d is even", n);
```

```
}
else
{
printf ("%d is odd", n);
}
getch();
}
```

(iii) what are the memory requirements for following data types in Turbo C 3.0 environment ?

a) int
b) char
c) float
d) double
e) void

=> a) int = 2 bytes
b) char = 1 bytes
c) float = 4 bytes
d) double = 8 bytes
e) Void = 0 bytes

(iv) what is explicit type conversion ? Explain with the help of a C program

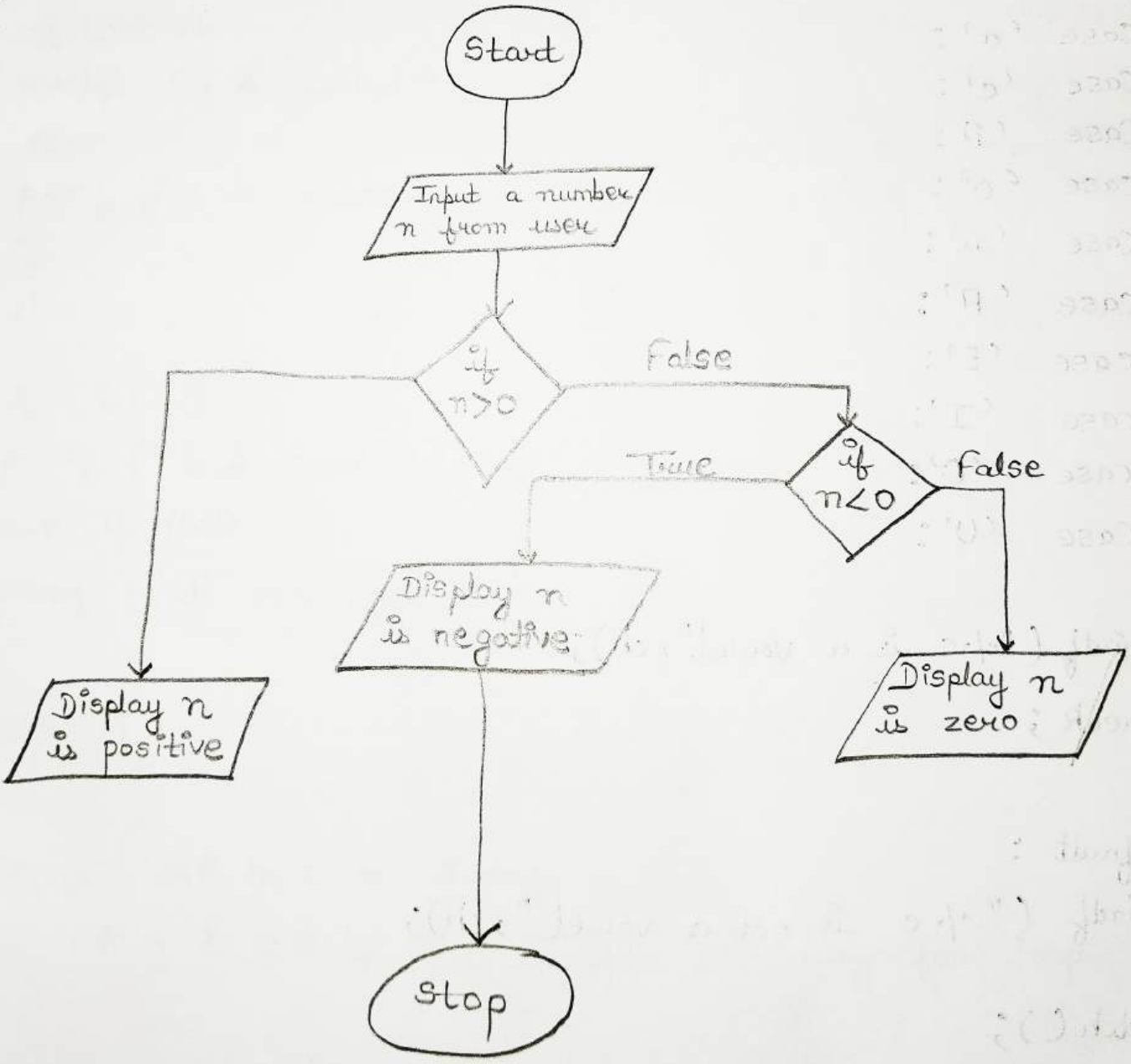
=> Explicit type Conversion in C is when the datatype conversion is user according to the program's need

=> Example :-

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

```
int a, b;
float c;
printf ("Enter value of a and b");
scanf ("%d%d", &a, &b);
  b = c;
  c = a * b;
printf ("%d multiply by %d is %d", a, b, c);
}
```

Q2. (a) Design a flowchart to check whether given input is positive, negative or zero



(b) WAP in C to check whether given input is vowel or not using ~~while~~ switch case

```
#include <stdio.h>
#include <conio.h>
Void main()
{
char ch;
clrscr();
printf("Enter a character");
scanf("%c", &ch);
Switch(ch)
{
Case 'a':
Case 'e':
Case 'i':
case 'o':
Case 'u':
case 'A':
case 'E':
case 'I':
case 'O':
Case 'U':
{
printf("%c is a vowel", ch);
break;
}
default:
printf("%c is not a vowel", ch);
}
getch();
}
```

(c) WAP to find greatest of three numbers without using logical operators

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int a,b,c;
clrscr();
printf("enter a,b,c");
scanf("%d%d%d", &a, &b, &c);
if (a>b)
{
if (a>c)
printf("a is greatest");
else
printf("c is greatest");
}
else
{
if (b>c)
printf("b is greatest");
else if (c<b)
printf("all are equal");
}
getch();
}
```

Q3. (a) WAP in c to display

1 * 3 * 5 * 7 ----- upto 51 * using for loop

```
=># include <stdio.h>
  # include <conio.h>
  Void main()
  {
  int i;
  clrscr();
  for (i=1; i<=51; i++)
  {
  if (i%2!=0)
  {
  printf ("%d*", i);
  }
  getch();
  }
```

(b) WAP to check whether input number is palindrome or not

```
=># include <stdio.h>
  # include <conio.h>
  Void main()
  {
  long int rev=0;
  int n, num, r;
  clrscr();
  printf ("enter the number");
  scanf ("%d", &n);
  num=n;
  while (n!=0)
{
r=n%10;
rev = rev* 10+r;
n = n/10
```

```
}
printf ("reverse of %d is %ld", num, rev);
if (num == rev)
printf ("\n %d is palindrome ", num);
else
printf ("\n %d is not a palindrome ", num);
getch ();
}
```

(c) WAP to display multiplication table of a number n

```
=># include <stdio.h>
  # include <conio.h>
  Void main()
  {
  int p, n, i;
printf ("enter the value of n");
scanf ("%d" , &n);
for (i=1; i<=10; i++)
{
    p = n* i;
printf ("%d*%d = %d", n, i, p);
}
getch ();
}
```

Q4. (a) Write an algorithm to calculate factorial of a number

=> Step 1:- Start

Step 2:- Declare a variable a and i

Step 3:- Input a number n from user

Step 4:- Initialize i=n and a= 1

Step 5:- i<=1

Step 6:- Cal. a = a * i

Step 7:- decrement i value by 1

Step 8:- repeat step 4

Step 9:- Print a

Step 10:- Stop

(b) Differentiate b/w while and do while loop with the help of an example

=> while loop checks the condition first and then executes the statement, whereas do while loop will execute the statement at least once, then the condition is checked while loop is entry controlled loop, whereas do while is exit controlled loop.

Example :-

=)//while

```
# include <stdio.h>
# include <conio.h>
 int main()
{
 int (num<=4)
{
 printf ("%d\n", num);
 num ++;
}
return 0;
}
```

```
// do while loop
# include <stdio.h>
# include <conio.h>
 int main ()
```

```
{
int main=1;
i=1;
do
{
printf("%d\n", 2* num);
num++;
}
while (num<=4);
return 0;
}
```

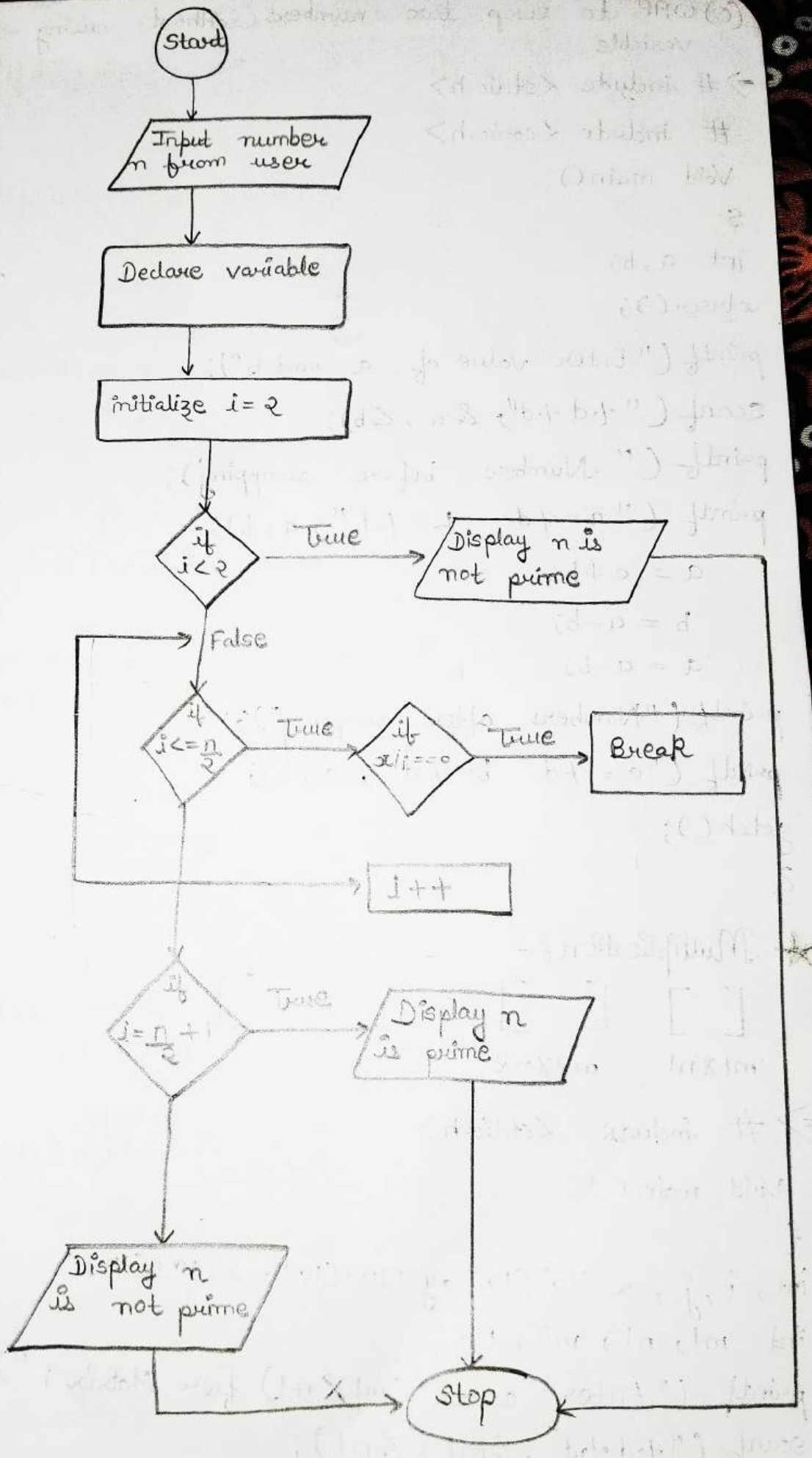
(c) WAP to display fibonacci series upto n terms

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int n, i, t1, t2, t;
printf("Enter no. of terms");
scanf("%d", &n);
printf("%d", t2);
for (i=2; i<=n ; i++)
{
t = t1 + t2;
printf("%d", t);
t1 = t2;
t2 = t;
}
getch ();
}
```

Q5. (a) WAP in C to find sum of digits of a number using while loop

```
#include <stdio.h>
#include <conio.h>
int main()
{
int num, sum=0;
clrscr();
printf("enter the number");
scanf("%d", &num);
while (num!=0)
{
  sum = num%10;
  num = num/10;
}
printf("sum of digits is %d", sum);
getch();
return 0;
}
```

(b) Draw a flowchart to check whether input number is prime number or not


Start
Input number n from user
Declare variable
initialize i = 2
if i<2
True
Display n is not prime
False
if i<=n/2
True
if n%i==0
True
Break
i++
if i=n/2+1
True
Display n is prime
Display n is not prime
Stop

(c) WAP to swap two numbers without using third variable

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int a, b;
clrscr();
printf("Enter value of a and b");
scanf("%d %d", &a, &b);
printf("Numbers before swapping");
printf("a=%d  b=%d", a, b);
    a = a+b;
    b = a-b;
    a = a-b;
printf("Numbers after swapping");
printf("a=%d  b=%d", a, b);
getch();
}
```

★ Multiplication :-

[ ] [ ] = [ ]

m1 x n1 m2 x n2

```
#include <stdio.h>
Void main()
{
int, i, j, x[10][10], y[10][10], z[10][10];
int m1, n1, m2, n2;
printf("Enter order (m1 x n1) for Matrix 1");
scanf("%d %d", &m1, &n1);
```

```
printf ("Enter order (m2 x n2) for Matrix 2");
scanf ("%d %d", &m2, &n2);
if (n1 != m2)
{
printf ("Multiplication not possible");
exit (1);
}
else
{
printf ("Enter elements for Matrix 1");
for (i=0; i<m1; i++)
{
for (j=0; j<n1; j++)
scanf ("%d", &x[i][j]);
}
printf ("Enter elements for Matrix 2");
for (i=0; i<m2; i++)
{
for (j=0; j<n2; j++)
scanf ("%d", &y[i][j]);
}
}
printf ("Resultant matrix is \n");
for (i=0; i<m1; i++)
{
for (j=0; j<n2; j++)
{
sum = 0;
for (k=0; k<n1; k++)
{
sum = sum + x[i][k] * y[k][j];
}
```

```
        z[i][j] = sum;
        printf ("%d", z[i][j]);
    }
    printf ("\n");
}
}
```

Strings...

Examples:-

```
#include <string.h>
#include <stdio.h>
Void main()
{
char name[50];
printf ("Enter your name");
scanf ("%s", name);
printf ("%s", name);
puts ("Enter another name");
gets (name);
puts (name);
}
```

Another example:-

```
#include <string.h>
#include <stdio.h>
Void main()
{
char name[] = "AIML2";
char name1[] = {'A', 'I', 'M', 'L', '2'};
printf ("size of name array is %d", sizeof (name));
printf ("size of name1 array is %d", sizeof (name));
}
```

→ String operations / Library functions in string.h :-

⇒ 1. Strlen (Str) ⇒ display length

```
# include <string.h>
# include <stdio.h>
Void main()
{
char x[] = "Hello";
int len = strlen(x);
printf("Length of the string is %d", len);
}
```

Strings...

1) Strlen (str)
2) Strrev
3) Strcpy (str1, str2)
4) Strcmp (str1, str2)
5) Strcat (str1, str2)
6) Strupr (str)
7) Strlwr (str)
8) Strcmpi (str1, str2)
9) Strncpy (str1, str2, n);
10) Strncat (str1, str2, n)

Example :-

```
# include <string.h>
# include <stdio.h>
Void main()
{
int i, count=0;
char x[30];
puts("Enter a string");
```

```
gets(x);
puts(" original string is ");
puts(x);
strrev(x);
puts("Reversed string is ");
puts(x);
}
```

2.
```
# include <stdio.h>
 Void main()
{
 int count, i;
 char x[30];
 puts("Enter a string");
 gets(x);
for (i=0; x[i] != '\0'; i++)
{
 count=1;
}
printf("No. of character as %d", count);
}
{
int count=0;
char x[30];
puts("Enter a string");
gets(x);
while (x[count] != '\0')
{
Count++;
}
printf("No. of characters %d", count);
}
```

3. # include <string.h>
include <stdio.h>
Void main()
{
char name [30], namerev [30];
int l, i, j=0;
printf ("Enter name");
gets (name);
l = strlen (name);
for (i= l-1; i>=0; i--)
{
printf ("%c", name [i]);
namerev [j] = name [i];
j++;
}
namerev [j] = '\0';
puts ("Reversed string is");
puts (namerev);
}

→ WAP to check whether input string is Palindrome or not

```
#include <string.h>
#include <stdio.h>
Void main()
{
char str [30], str1 [30];
printf ("Enter the string");
gets (str);
strcpy (str1, str);
strrev (str1);
if (strcmp (str1, str)==0)
printf ("string %s is a palindrome", str);
else
printf ("string %s is not a palindrome", str);
}
```

→ Strcat (Str1 , Str2) :-

⇒ # include <string.h>
include <stdio.h>
Void main ( )
{
char Str1 = "Hello";
char Str2 = "Rupit";
Strcat (Str1 , Str2);
printf (" strings after concatenation is %s", str1);
}

Functions...

⇒ A function is a group of statements that performs a specific task and is relatively independent of the remaining code.

Function are used to organise program into smaller and independent unit.

→ Advantages :-

⇒ 1. Reduction in code Redundancy.
2. Enable code reuse.
3. Better readability.
4. Information hiding.
5. Improved debugging and Testing.
6. Program becomes easy to manage.

→ Difference b/w user-defined function and library function.