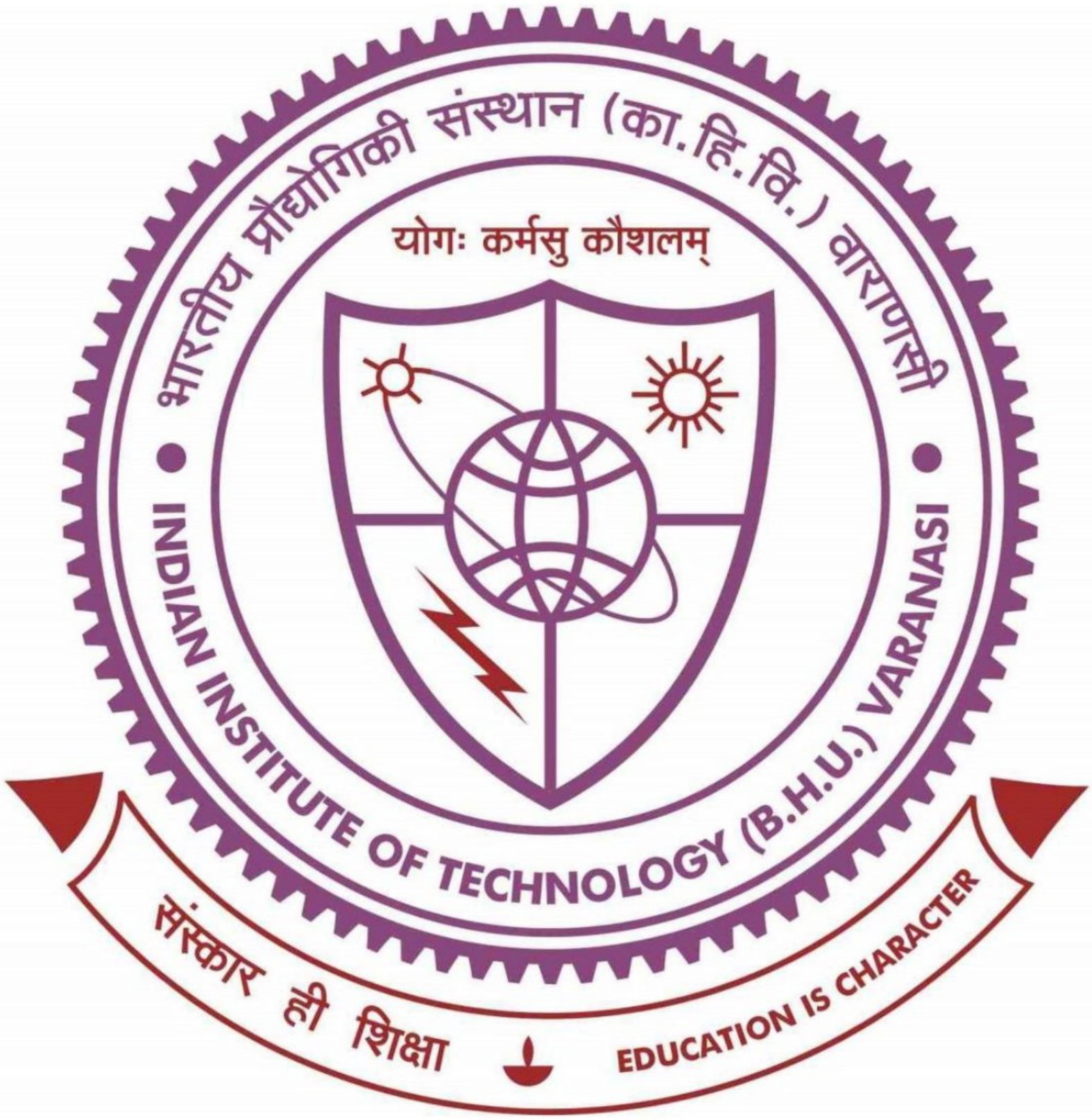


NAME-PARTIK SINGH BUMRAH

ROLL NO-21075064

EMAIL ID - partik.sbumrah.cse21@itbhu.ac.in

DEPARTMENT-CSE

SUBJECT- CSO-101

SECTION-BA-1

1. In the question there will be an error as the data type of n and y is not defined.

But if we consider that the data type is int then the value of n = -10.

2. int n=6, y=-8;

n* = n%y ? n-y ? -- n : y++;

the value of n after execution of this program will be.

n=-48;

3. #include <stdio.h>
int main()
{
int n, y=5, z=5;
printf("%d", n);
getchar();
return 0;
}

The output of this program will be 1 as "==" is a relational operator used to check the equality of two values and it returns

1. 1 if equality holds.

2) 0 if equality does not hold.

And as in this case $y = 3 > 5$ so $x = 1$.

And the output will be displayed after the user press enter as getchar() is an unformatted input function which requires to press enter to execute.

4) TRUE

```
int main() {

    int i = 3;
    printf("%d", (++i)++);

    return 0;

}
```

The output of this program will be compilation error as in C, the pre-increment or post-increment operators are used on L-value expression. And as (++i). is an R-value expression which cannot be post incremented. Hence the compiler throws an error.

5) Expr-1 = 8

Expr-2 = 65.000000

Expr-3 = 1.000000.

6) $x * w + y * z = 7$

will have one pair of parentheses

so the correct equation should be..

$x * w + y * (z = 7)$

if $x = 2$, $y = 3$, $z = 5$, $w = 7$

so ~~of~~ evaluating

$2 * 7 + 3 * (7) = 14 + 21 = 35$ Ans

so corrected Expression = $x * w + y * (z = 7)$

and. Value of the corrected expression = 35

Date
Page

7) Registers are very fast computer memory which are used to execute programs and operations efficiently. The purpose of having registers is just retrieve of data for processing by CPU. All computers load data from a large memory into registers where it is used for arithmetic operations and is manipulated or tested by machine instructions. Manipulated data is then often stored back to main memory, either by same instruction or by a subsequent one.

There are a lot of types of registers

1) Accumulator

Most ~~recently~~ frequently used register used to store data taken from memory. It is in different numbers for different microprocessors.

2) Memory Address Registers (MAR)

The address of the location which is to be accessed from memory is contained in MAR, also it is responsible for the communication of CPU and the main memory.

3) Memory Data Registers (MDR)

The MDR contains the data that needs to be written or read out from the addressed location of the main memory.

4) General Purpose Registers

These registers are used to store temporary data during any ongoing operation. The content of these registers can be accessed by assembly programming.

5) Program Counter (PC)

This register contains the address of the next instruction to be fetched of the main memory.

6) Instruction Register (IR)

The IR holds the instruction which is just about to be executed.

7) Condition Code Register (CCR)

CCR show the status of any operation through different flags.

8) A number can be printed in Hexadecimal and octal in C by using format specifiers. And the format specifier for printing a hexadecimal number is %.x and for octal we can use %.o format specifier.

For Eg:-

```
int y = 5000;
printf("%.x  %.o", y, y);
```

This program will print decadecimal and octal of y=5000.

The print statement for printing 123.331, 123.3, 123.
, 1.233 e+02

```
float y = 123.331;
printf("%.3f \n", y);
printf("%.1f \n", y);
printf("%.2f \n", y);
printf("%.3e \n", y);
```

Q. Difference between formatted and unformatted input?

Formatted	Unformatted
a) Formatted input is in user desired format.	a) Unformatted input is the most basic form of input and is not in user desired format.
b) Formatted input can be given through some format specifiers in syntax.	b) Unformatted input doesn't need some format specifier in their syntax.
c) Formatted input is data is more user friendly way	c) Unformatted input uses less space for storing data.
d) Formatted input is used for all data types.	d) Unformatted input is mainly used for character and string data types.

Some formatting characters in C are

- %c for character
- %d for signed integer
- %e for scientific notation
- %f for float values.

A scanf function to parse input on # sign and read two numbers is:

```
int x, y;
scanf("%d#%d", &x, &y);
```

Eg:- ~~printf("%d~~ If the input is 124#144 then x=124 will be stored in x and 144 will be stored in y

10. A for loop is used to execute a code for a specific number of times.

Syntax of for loop

```
For (data type i= ; i< count, i++).
{
    code to be executed
}
```

In for loop a loop variable is used to control the loop which needs to be initialized at start., i is the loop variable in case of the example

There are three expressions in for loop.

1) Initialization Expression - In this expression we have to initialize the loop variable. For example: int i=0;.

2) Test expression - In this expression we have to test the condition. If the condition is true then the code in for loop is executed if the condition is false then the code is not executed and the loop is exited.

3) Update Expression - After executing the loop the loop variable is updated.

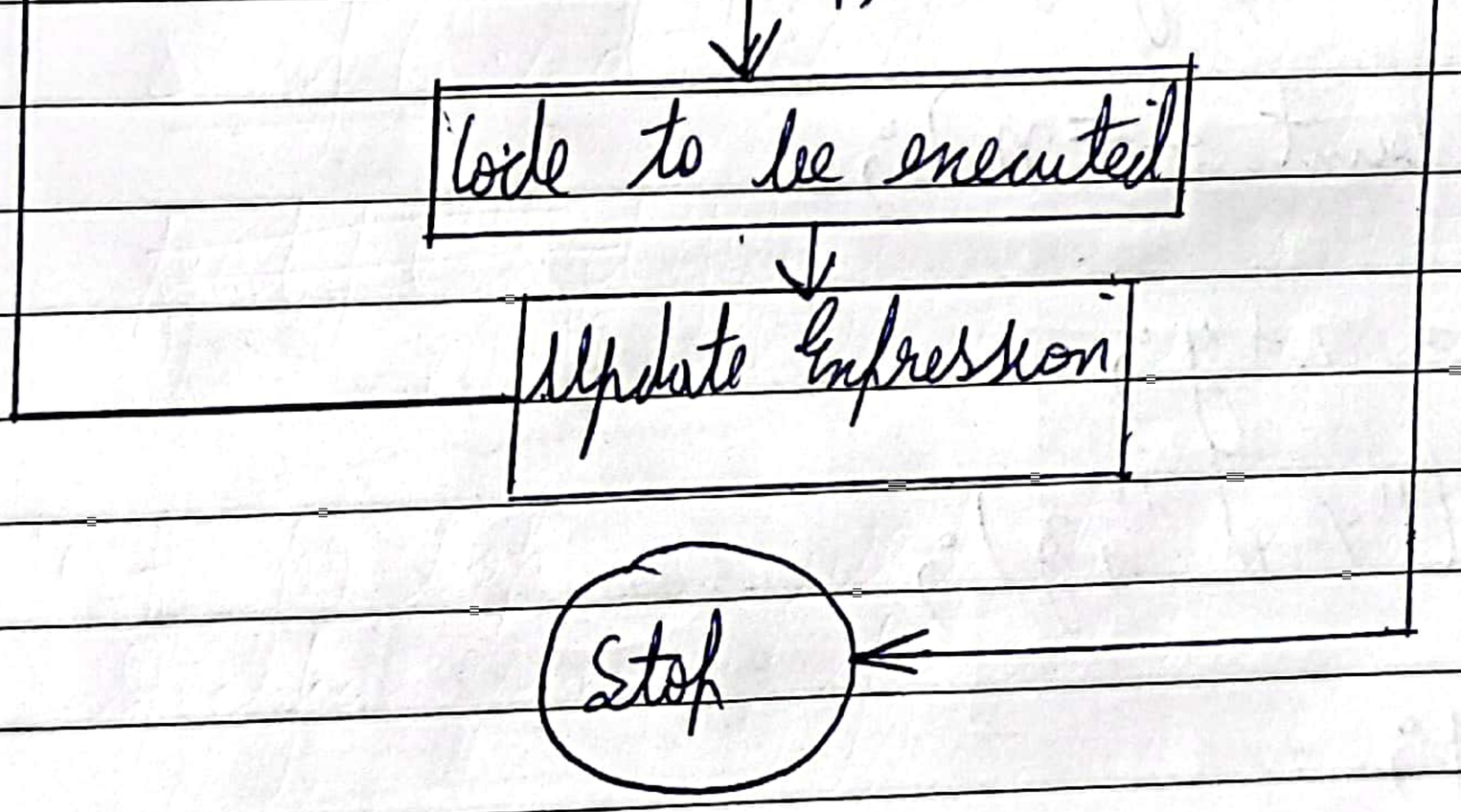
And this process is repeated until the test conditions fails.

2) The difference between For loop and while loop

For	while
1) It is a counter controlled loop.	1) It is a sentinel-controlled loop
2) It is normally used when number of times the loop needs to be executed is known.	2) It is used when the condition for which the loop needs to be executed is known.
3) Condition is a relational expression	3) Condition may be expression or non-zero value.

4.6 For loop is an entry controlled loop.	4.7 While loop is also entry controlled loop

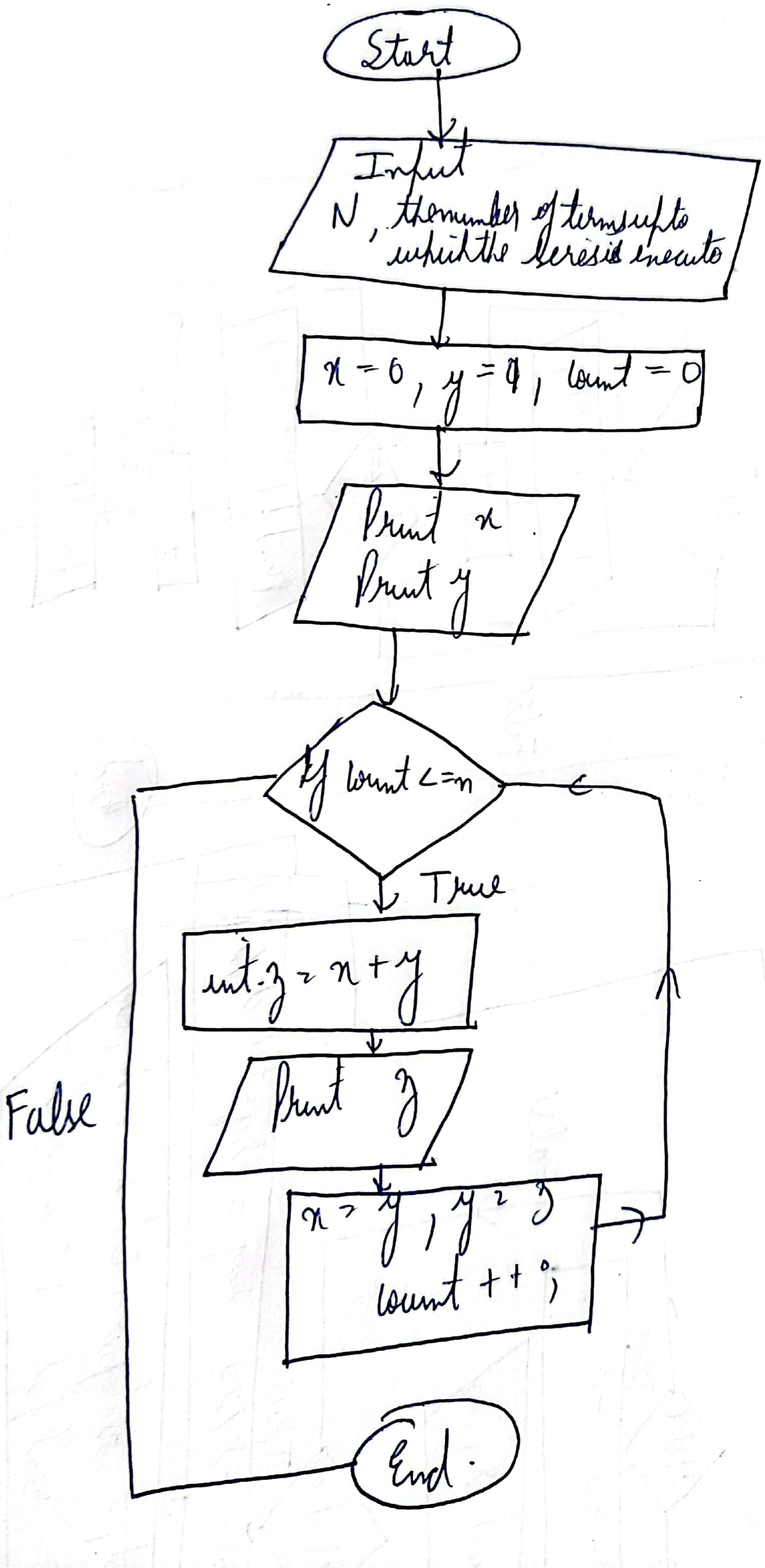
Flow chart of For loop



11. A program for the series

0, 1, 1, 2, 3, 5, 8, 13, 21.

```
#include <stdio.h>
int main()
{
    int x=0, y=1, n, count=0;
    printf("Enter upto how many terms you want the
            series to be printed \n");
    scanf("%d", &n);
    printf("%d", x);
    printf("%d", y);
    while (count <=n);
    {
        int z = x+y;
        printf("%d", z);
        x = y;
        y = z;
        count++;
    }
    return 0;
}
```


Start
Input
N, the number of terms upto which the Series is execute
x = 0, y = 1, count = 0
Print x
Print y
If count <= n
True
False
int z = x + y
Print z
x = y, y = z
count ++;
End.

The smallest individual units are known as C tokens. These individual units that are meaningful to the compiler.

Different types of tokens are :—

1. Keywords - Keywords are reserved words in a programming language. Each keyword is reserved to perform a specific function in a program. These words can't be used as variable names. C language has 32 keywords

2. Identifiers - Identifiers are used for the naming of variables, functions and arrays, these are user-defined names.

3. Constant - These are the values that remains fixed during the execution of the program once they are defined. Constants may belong to any of the data type.

4. Strings - Strings are an array of characters ending with a null character ('\0').

5. Special Symbols - These are some symbols having some special meaning. Eg :- [], {}, ;, (), =, #.

6. Operators - Operators are symbols that execute an action when applied to C variables and other objects.

13. Pre processors are not part of the compiler, but is a separate step in the compilation process. Pre processor directives such as # define, are typically used to make source programs easy to change and easy to compile in different execution environments.

Date
Page

The functions of a pre processor.

1) Removing comments - It removes all the comments and as they are not required in the execution and won't be executed as well

2) File inclusion - Including all the files from library that our program needs. # include is a directive for the preprocessor that tells us to include the contents of the library file

14) We use # include directive to include contents of another file to a program.

We use # include in two different ways

a) # include < file . h >

This variant is used to include a header file. The Standard header file directory is the path where all header files are stored.

b) # include " file "

We use this variant to include our own / custom header files.

We use # define to define a name for particular value / constant / expression.

12. Int main ()	1. Void main ()
1. It is used to define a main function which takes no argument and returns an integer data type.	1. It is used to for undefined is reserve of the main function and it can throw garbage results or an error sometimes.

C. Imperative Programming	Declarative Programming
1. In this, programs specify how it is to be done.	1. In this, programs specify what is to be done
2. It simply describes the control flow of computation	2. It simply expresses the logic of computation.
3. Its type include procedural programming, object-oriented programming etc.	3. Its type include logic programming and functional programming.

15. Nested loop basically means a loop statement inside another loop statement. Any number of loops can be defined another loop. And we can define any type of loop inside any other type of loop.

Syntax for nested loop

Outer — loop:

{ Inner — loop:

{

code 1

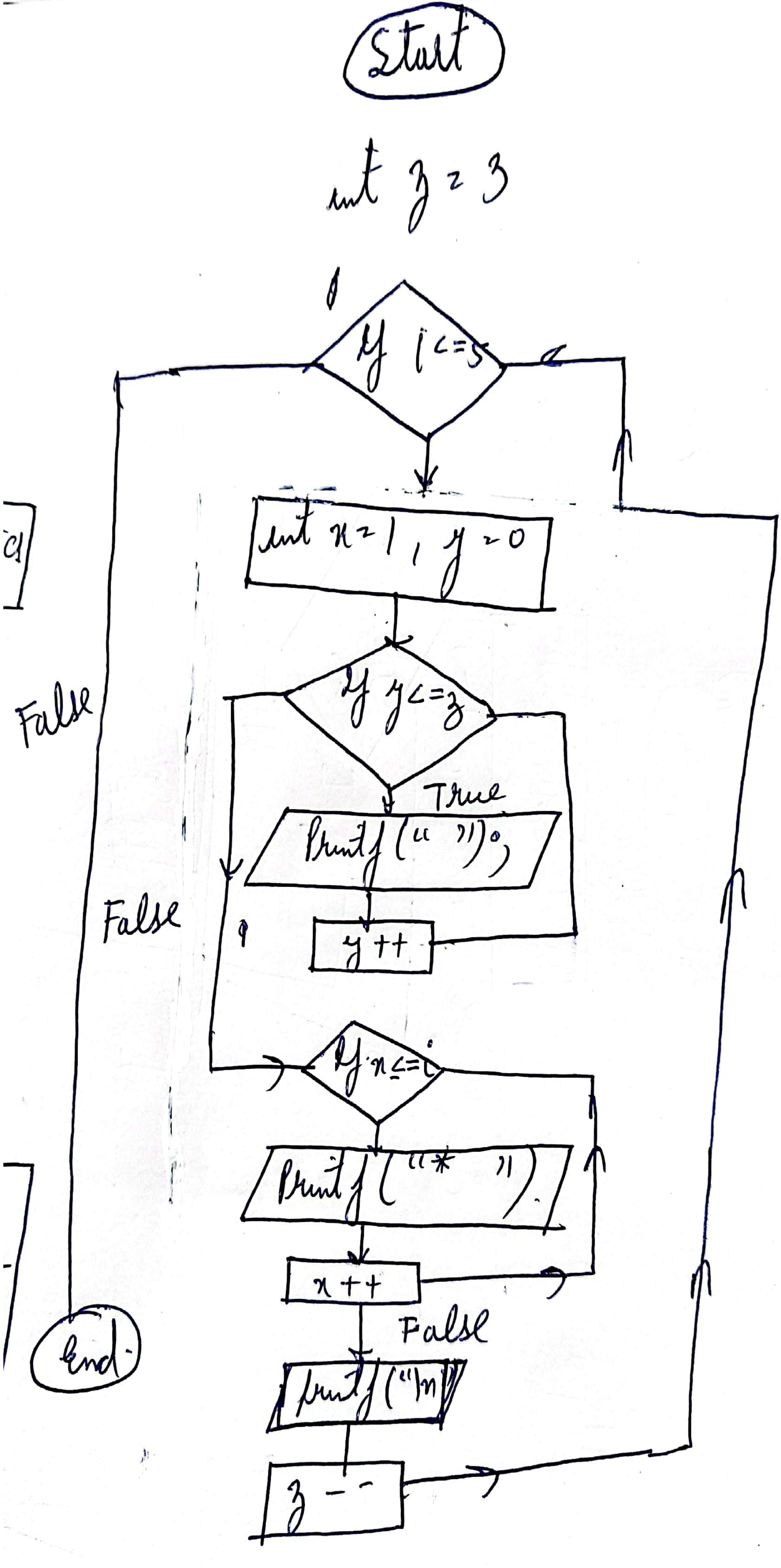
}

code 2

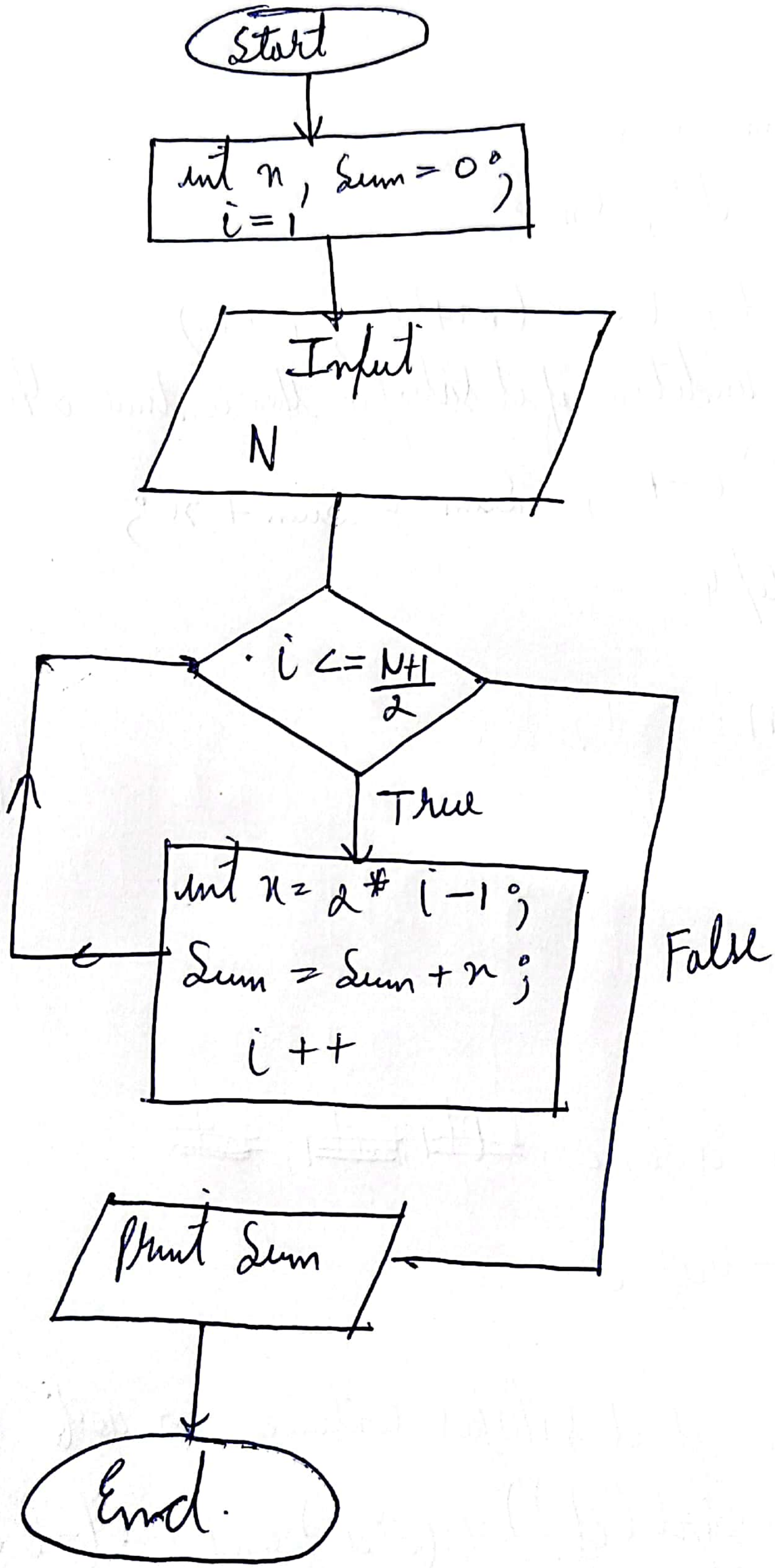
}

Algorithm

Step 1	Start
Step 2	$z = 3$;
Step 3	For (i=1, i<=5, i++)
Step 4	Check for condition if it satisfies go to next step, or go to step
Step 5	x = 1, y = 0
Step 6	while (y <= z)
Step 7	Check for condition if it satisfies go to next step, or go to step 9
Step 8	printf (" "); , y ++
Step 9	while (x <= i)
Step 10	Check for condition if it satisfies go to next step, or got to step 12
Step 11	printf ("* "); , x ++;
Step 12	printf ("\n"); , z--;
Step 13	~~Step~~ go to step 3
Step 14	Stop


start
int z=3
for(i=1; i<=5; i++)
int x=1, y=0
for(y=1; y<=z; y++)
printf(" ");
y++
for(x=1; x<=i; x++)
printf("* ");
x++
printf("\n");
z--
end
True
False
False
False

Step 1	Start
Step 2	n , Sum = 0 ;
Step 3	Scanf ("%d", &n);
Step 4	For (i=1 , i <= (n+1)/2 ; i++)
Step 5	Check for condition if it satisfies then continue otherwise go to step 9.
Step 6	int x = 2*i-1 ; Sum = Sum + x ;
Step 7	go to step 4.
Step 8.	print sum.
Step 9	stop


Start
int n, Sum = 0;
i = 1;
Input
N
i <= N+1 / 2
True
int x = 2 * i - 1;
Sum = Sum + x;
i++
False
Print Sum
End.

Step 1 Start

Step 2 Take input for a, b, c, ~~root1~~, ~~root~~, ~~root2~~

Step 3 $d = b*b - 4a*c$

Step 4 If $(d > 0)$

Step 5 check condition if it satisfies continue or goto step 8

Step 6 root 1 = $(-b + sqrt(d)) / (2*a)$, root 2 = $(-b - sqrt(d)) / (2*a)$.

Step 7 ~~print~~ print root 1 root 2

Step 8 If $(d == 0)$

Step 9 check the condition if it satisfies continue or go to step

Step 10 $root_1 = root_2 = -b / 2*a$;

Step 11 print root 1, root 2.

Step 12 If $(d < 0)$

Step 13 check condition if it satisfies continue or go to step

Step 14 real part = $-b / 2*a$; img part = $sqrt(-d / 2*a)$;

Step 15 print $root_1 =$ realpart + imgpart, root 2 = realpart - imgpart;

Step 16 Stop.

17.
```
#include <stdio.h>
int main()
{
    int age, balance;
    float interest, penalty;
    char ch;
    printf("Enter age and balance\n");
    scanf("%d %d", &age, &balance);
    printf("Enter gender M for male and F for female\n");
    scanf("%c", &ch);
    if (age >= 60)
    {
        if (balance >= 75000)
        {
            interest = (3.25 * balance) / 100;
            penalty = 0;
        }
        else
        {
            interest = (3.25 * balance) / 100;
            penalty = 0;
        }
    }
```

```
else if (ch == 'F')
{ if (balance >= 100000)
  { interest = (4.75 * balance) / 100;
    penalty = 0;
  }
}
else if (balance <= 5000 && age < 60)
{ interest = (3.75 * balance) / 100;
  penalty = (1.25 * balance) / 100;
}
else {
    interest = (3.75 * balance) / 100;
}
printf ("%.d\n", interest);
printf ("%.d", penalty);
return 0;
```

```
#include <stdio.h>
int main()
{
int m1, m2, m3, m4, m5, m6, m7, m8, m9, m0;
m1 = m2 = m3 = m4 = m5 = m6 = m7 = m8 = m9 = m0 = 0;
int x;
while (1)
{
scanf("%d", &x);
if (x == -1)
{
goto label;
}
else
{
while (x != 0)
{
int y = x % 10;
Switch(y)
{
case 0: m0++; break;
case 1: m1++; break;
case 2: m2++; break;
case 3: m3++; break;
```

```
case 4: m4++; break;
case 5: m5++; break;
case 6: m6++; break;
case 7: m7++; break;
case 8: m8++; break;
case 9: m9++; break;
}
x = (x-z)/10;
}
}
label:
printf("0: %d\n", m0);
printf("1: %d\n", m1);
printf("2: %d\n", m2);
printf("3: %d\n", m3);
printf("4: %d\n", m4);
printf("5: %d\n", m5);
printf("6: %d\n", m6);
printf("7: %d\n", m7);
printf("8: %d\n", m8);
printf("9: %d\n", m9);
return 0;
}
```