

NAME-PARTIK SINGH BUMRAH

ROLL NO-21075064

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

DEPARTMENT-CSE

SUBJECT- CSO-101

SECTION-BA-1

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

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

2.8 int $x=6, y=8;$

$n^* = n \% y > n-y ? --n : y++;$

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

$n = -18;$

3. #include <stdio.h>
int main()

{

int $x, y=5, z=5;$

printf("%d", x);

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 if equality holds.

2) 0 if equality does not hold.

And as in this case $y = z^2 - 5 \leq n - 1$.

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

4) ~~Ans~~

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 expressions. 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

$$6) \quad x^w + y^z = ?$$

will have one pair of parentheses

so the correct equation should be :-

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

$$\text{if } x=2, y=3, z=5, w=7$$

to ~~if~~ evaluating

$$2^w + 3^z(z=7) = 14 + 21 = 35 \text{ Ans} =$$

$$\text{so corrected expression} = x^w + y^z(z=7)$$

and Value of the corrected expression = 35

7) Registers are very fast computer memory which are used to execute programs and operations efficiently. The purpose of having registers is fast retrieval 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 a 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)

CCF 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 z = 5000 ;

printf ("%x %o", z, z);

This program will print denadecimal and octal of 3^{2500} .

The print statement for printing 123.331, 123.3123,
, 1.233 E+02

8) 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 specifier.

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 integers
- ✓ %e for scientific notation
- ✓ %f for float values.

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

int n, y;

sscanf ("% .d # % .d" , &n, &y);

Eg:- ~~printf ("")~~ If the input is 124 # 144
then 124 will be stored in n 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 (datatype i = ; i < count, i++).

Code to be executed

4

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 condition fails.

The difference between For loop and while loop

For

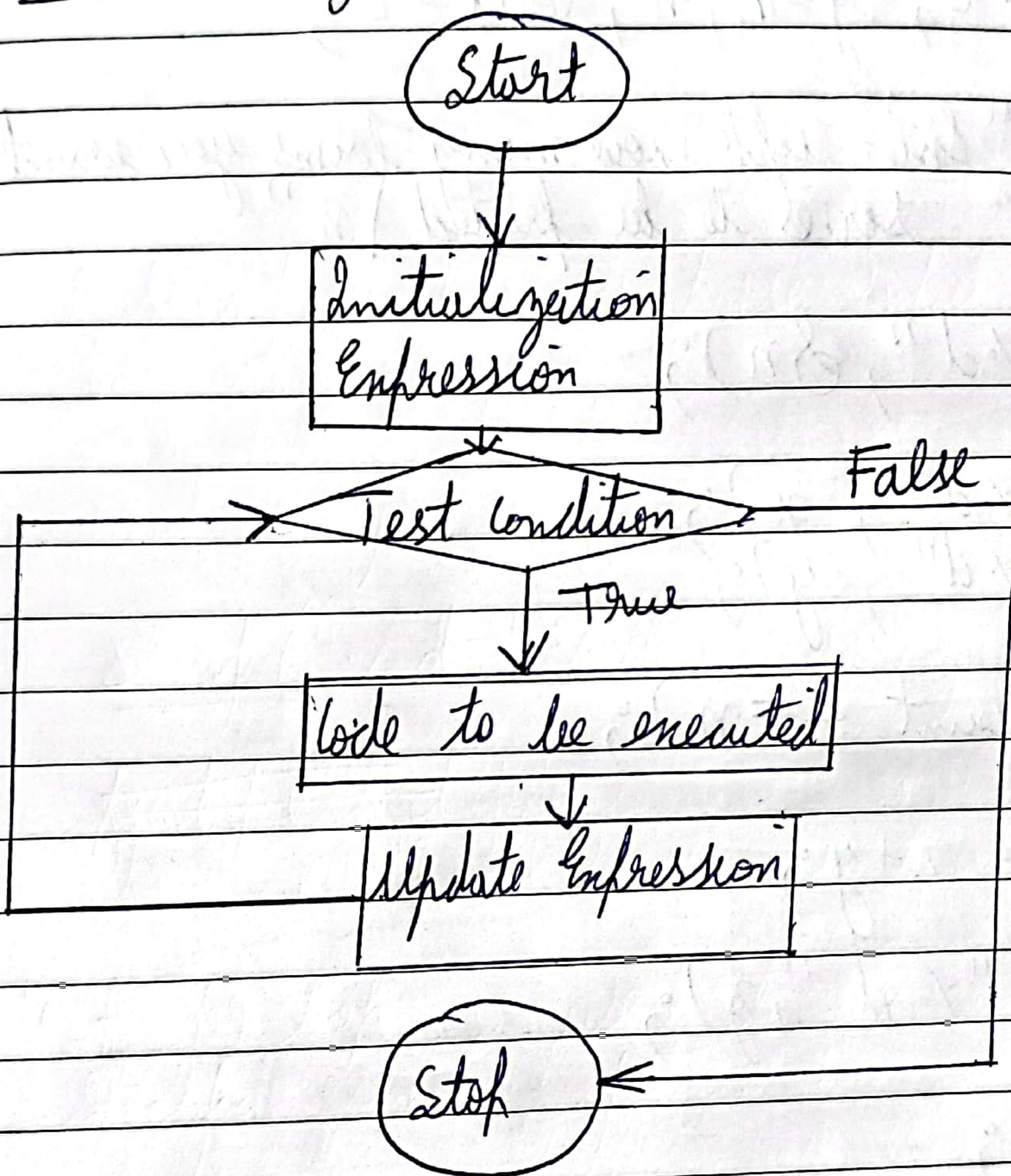
while

- | | |
|---|--|
| 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 non-zero value. |

4.4 For loop is an entry controlled loop.

4.5 While loop is also entry controlled loop.

Flow chart of For loop



11.4 A program for the series

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

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

```
{ int n=0, y=1, x, 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);
```

$x = y;$

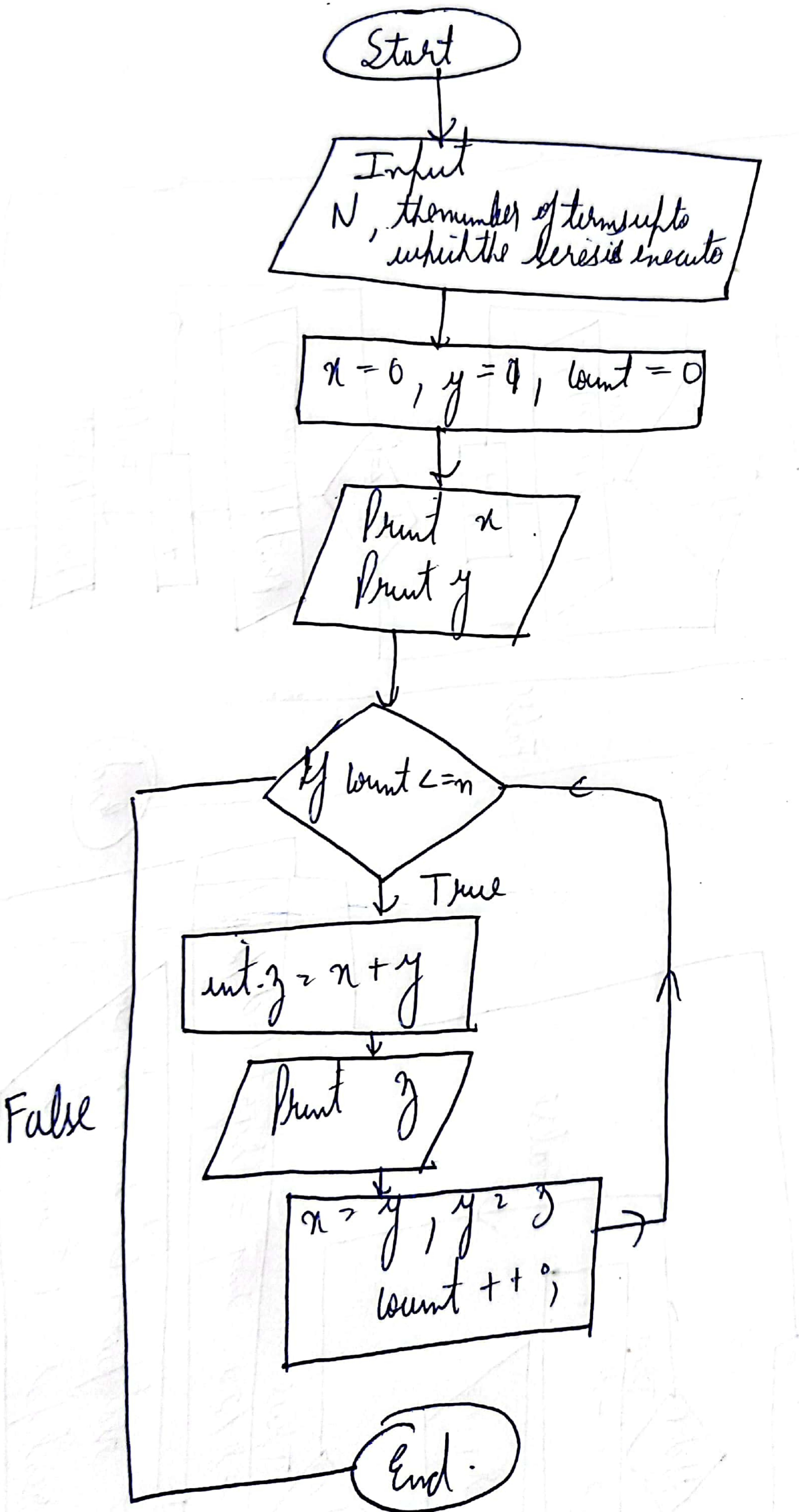
$y = z;$

count ++;

}

return 0;

6



The smallest individual units are known as 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. (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.
7. 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.

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 it to include the contents of the library file.
- 3) 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. 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.

1. Int main ()

1. It is used to define a main function which takes no arguments and returns an integer data type.

2. Void main ()

1. It is used to for undefined usage of the main function and it can throw garbage results or an error sometimes.

C. i) Imperative Programming

1. In this, programs specify how it is to be done.

2. It simply describes the control flow of computation.

3. Its type include procedural programming, object-oriented programming etc.

ii) Declarative Programming

1. In this, programs specify what is to be done

2. It simply expresses the logic of computation.

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

of code 1

}

code 2

{

Algorithm

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

Start

int z = 3

a)

False

False

False

End

if i <= 5

int n = 1, y = 0

if y <= z

True

printf(" ") ;

y++

if n <= i

printf("* ") ;

x++

printf("n")

z--

Step 1

Step 2

Step 3

Step 4

Step 5

Step 6

Step 7

Step 8

Step 9

Start

$n, \text{sum} = 0;$

Scarf ("r.d", $\delta_n);$

For (. $i \geq 1, i <= (n+1)/2, i++$)

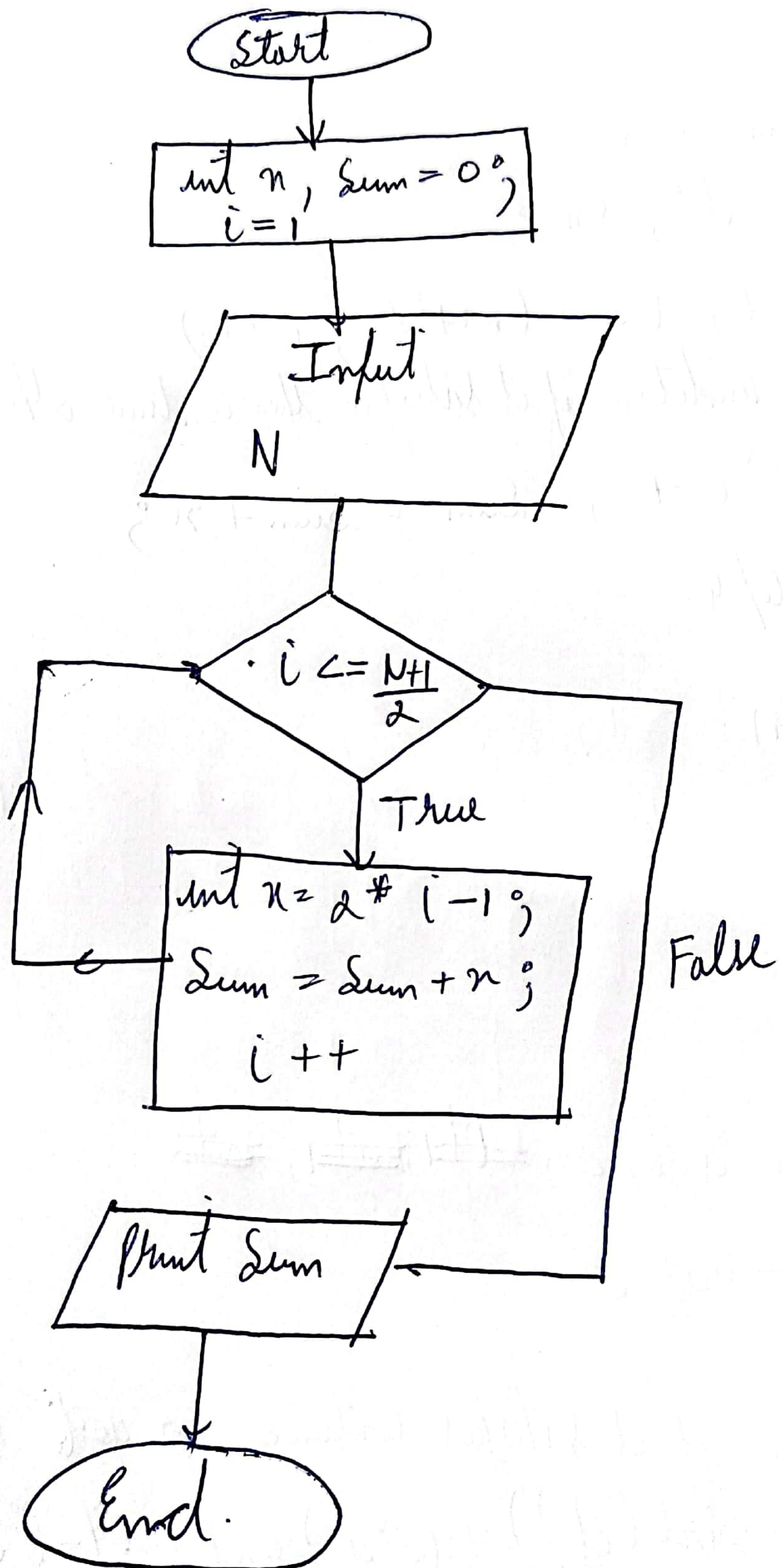
Check for condition if it satisfies then continue otherwise go to step 9.

int $n = 2 * i - 1, \text{sum} = \text{sum} + n;$

Go to step 4.

print sum.

Stop



Step 1

Step 2

Step 3

Step 4

Step 5

Step 6

Step 7

Step 8

Step 9

Step 10

Step 11

Step 12

Step 13

Step 14

Step 15

Step 16

Start

Take input for $a, b, c, \text{root1}, \text{root2}$

$$d = b^2 - 4ac$$

If ($d > 0$)

check condition if it satisfies continue or go to step 8

$$\text{root1} = \frac{-b + \sqrt{d}}{2a}, \text{root2} = \frac{-b - \sqrt{d}}{2a}$$

~~print~~ root1 root2

If ($d == 0$)

check the condition if it satisfies continue or go to step

$$\text{root1} = \text{root2} = \frac{-b}{2a};$$

~~print~~ root1, root2

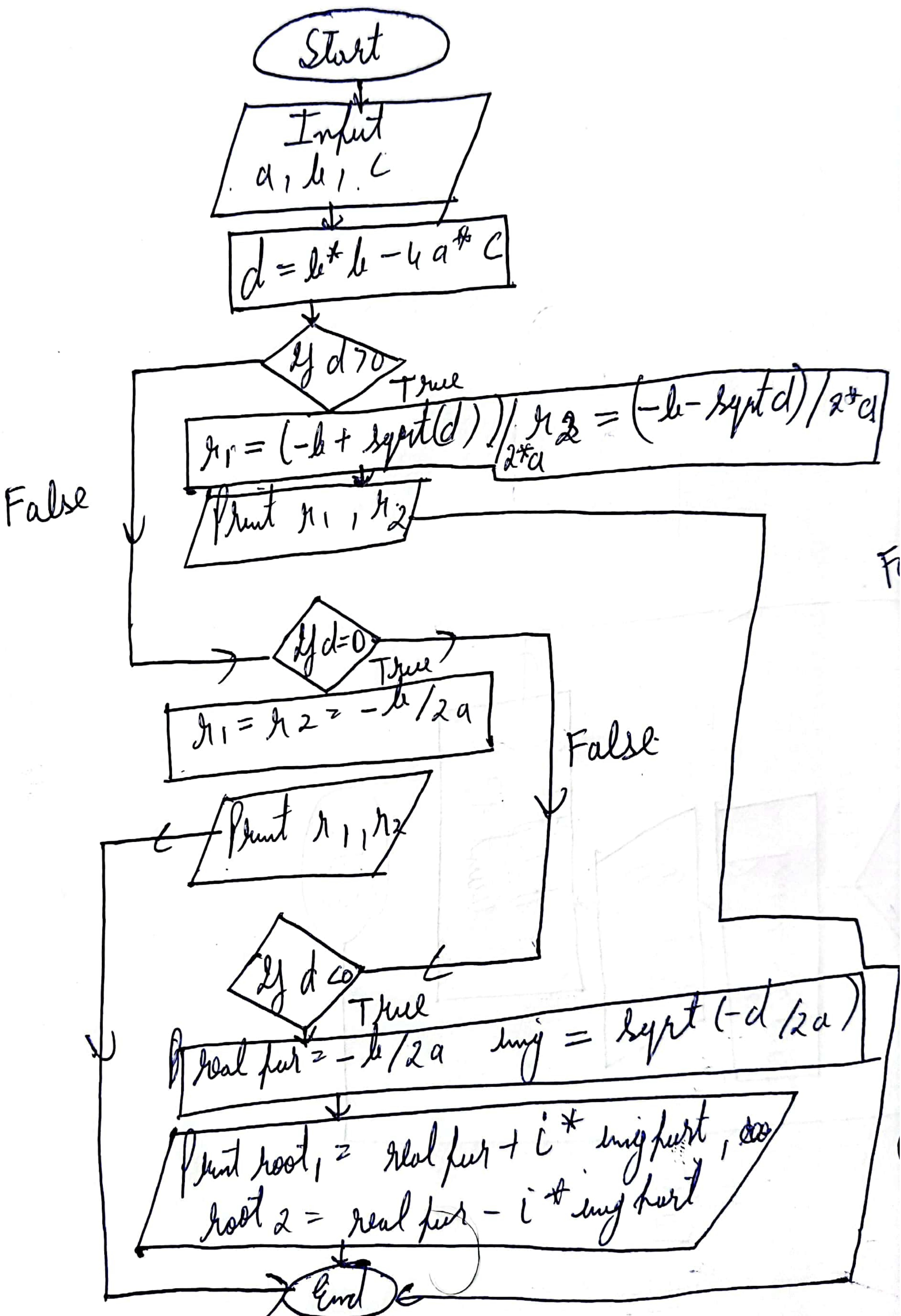
If ($d < 0$) check condition if it satisfies continue or go to stop

check condition if it satisfies continue or go to stop

real part = $-b / 2a$; img part = $\sqrt{-d / 2a}$,

print $\text{root1} = \text{realpart} + i \text{imgpart}$, $\text{root2} = \text{realpart} - i \text{imgpart}$,

stop.



17. # include <stdio.h>

int main ()

{ int age, balance ;

float interest, penalty ;

char ch ;

printf ("Enter age, and balance") ;

scanf ("%d %f", &age, &balance) ;

printf ("Enter gender M for male and F for female") ;

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 ("%.2f", interest);

printf ("%.2f", penalty);

return 0;

8. #include <stdio.h>
int main ()

{ int ~~m1~~ = ~~m2~~; int

int $m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}$

$m_1 = m_2 = m_3 = m_4 = m_5 = m_6 = m_7 = m_8 = m_9 = m_{10} = 0$

int x° ;

while (1)

{ scanf ("%d", & x°);

if ($x^{\circ} == -1$)

{ goto label;

}

else

{ while ($x^{\circ} != 0$)

{

int $y = x^{\circ} \cdot 10^{\circ}$;

switch (y)

{ case 0: m_0++ ; break;

case 1: m_1++ ; break;

case 2: m_2++ ; break;

case 3: m_3++ ; break;

Call 4 : m₄++ ; break ;
Call 5 : m₅++ ; break ;
Call 6 : m₆++ ; break ;
Call 7 : m₇++ ; break ;
Call 8 : m₈++ ; break ;
Call 9 : m₉++ ; break ;

p

$$n = (n - 3) / 10 ;$$

p

p

label :

```
printf ("0 : -f d \n", m0);  
printf ("1 : -f d \n", m1);  
printf ("2 : -f d \n", m2);  
printf ("3 : -f d \n", m3);  
printf ("4 : -f d \n", m4);  
printf ("5 : -f d \n", m5);  
printf ("6 : -f d \n", m6);  
printf ("7 : -f d \n", m7);  
printf ("8 : -f d \n", m8);  
printf ("9 : -f d \n", m9);
```

return 0;

p