Experiment No. 01

Aim: Study of Operators in C.

Problem Statement: Write a program to accept the temperature in Celsius and to convert anddisplay it in Fahrenheit.

Problem Definition:

Input: Value of temperature in Celsius

Processing: Convert temperature in Celsius to Fahrenheit.

$$F=1.8*C+32$$

Output: Value of temperature in Fahrenheit.

Theory:

An operator is a symbol that specifies the mathematical, logical or relational operator to be performed C language supports following type of operators.

- Arithmetic Operators
- Logical (or Relational) Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators:
There are following arithmetic operators supported by C language: Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10

*	Multiply both operands	A * B will give 200
/	Divide numerator by denumerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Logical (or Relational) Operators:
There are following logical operators supported by C

language Assume variable A holds 10 and variable B

holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition	(A < B) is true.

	becomes true.	
>=	Checks if the value of left operand is greater than or equal to the value of	(A >= B) is not true.

	right operand, if yes then condition becomes true.	
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Bitwise Operators:

Bitwise operator works on bits and performs bit by bit operation.

Assume if A = 60; and B = 13; Now in binary format they will be as follows: A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

There are following Bitwise operators supported by C language

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators:

There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigned value of A + B into C

+=	Add AND assignment operator, It adds right operand to the left	C += A is equivalent to C = C + A

	operand and assign the result to left operand	
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2

|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Misc Operators

There are few other operators supported by C Language.

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is interger, will return 4.
&	Returns the address of an variable.	&a; will give actaul address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Categories:

All the operators we have discussed above can be categorised into following categories:

- Postfix operators, which follow a single operand.
- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.
- The comma operator, which guarantees left-to-right evaluation of comma-separated expressions.

Precedence of C Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedenace than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedenace operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * &sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<<>>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Algorithm:

Step1: Start

Step2: Accept the temperature in Celsius

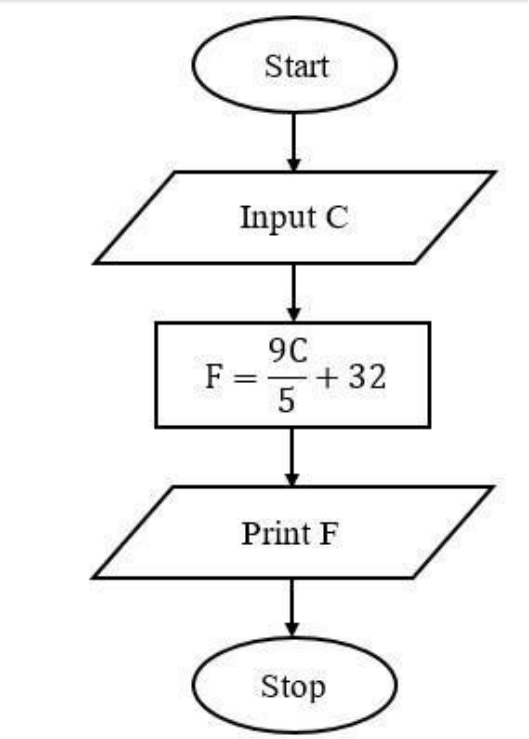
Step3: Calculate temperature in

Fahrenheit F=9/5 *C+32

Step4: Display temperature in Fahrenheit

Step5: Stop

Flowchart:



Source Code:

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

void main ()
{
        float f, c;
        clrscr();
        printf("Enter Temperature in Celsius:");
        scanf("%f",&c);
        f=9.0/5*c+32;
        printf("Temperature in Fahrenheit:%f",f);
        getch();
}
```

Output:

```
        Enter Temperature in Celsius:
        30 Temperature in
        Fahrenheit:86
```

Experiment No. 02

Aim: Study of if-else structure in 'C'

Problem Statement: Write a program to accept three numbers and display largest of three using a nested if else statement

Problem Definition:
Input: Accept three numbers
Processing: Find the maximum of three numbers
Output: Display maximum number

Theory:
The if statement
The if statement gives the user the choice of executing a statement (possibly compound) if the expression is evaluated to true or skipping it is the expression is evaluated to false.

Format 1:

```
if (expression)
{
        statement
}
```

The *statement* is executed if and only if the *expression* is true.
Example

```
if (num > 10)
{
    result = 2 * num;
}
```

The content of *num* is multiply by 2 if and only if the value of *num* is greater than 10.

Format 2: C language also lets one choose between two statements by using the if-else if structure.

```
if (expression)
{
statement 1
}
else
{
    statement 2
}
```

In this case, if the *expression* is true, then the *statement 1* is executed. Otherwise,*statement 2* is executed.

Example

```
if (num > 10)
{
                                        result = 2 * num;
}
else
{
                                        result = 3* num;
}
```

In the above example, if the *num* is greater than 10 then the *result* is equal to the *num* multiplied by 2 otherwise it is equal to the *num* multiplied by 3.

Algorithm:

Step1: Start

Step2: Accept three numbers a, b, c

Step3: if (a>b) then

if(a>c) then

display a

else

display b

else

if(b>c) then

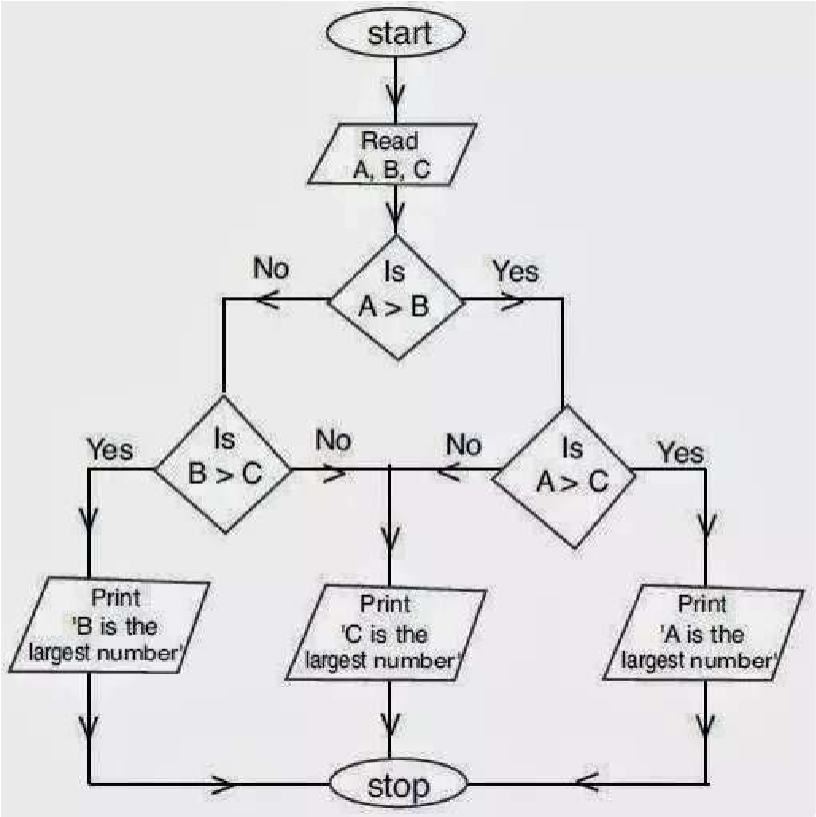
display b

else

display c

Step5: Stop

Flowchart:


start
Read
A, B, C
Is
A > B
No
Yes
Is
B > C
Yes
No
No
Is
A > C
Yes
Print
'B is the
largest number'
Print
'C is the
largest number'
Print
'A is the
largest number'
stop

Source Code:

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

void main ()
{
        int a, b, c;
        clrscr();
        printf("Enter three numbers");
        scanf("%d %d %d", &a, &b, &c);
        if(a>b)
        {
                        if(a>c)
                        {                       printf("%d is the largest
                                                number", a);
                        }
                        else
                        {
                                                printf("%d is the largest
                        }                       number", c);
        }
        else
        {
                        if(b>c)
                        {

                        }                       printf("%d is the largest
                        else                    number", b);
                        {

                        }
                                                printf("%d is the largest
                    }                           number", c);
                getch();
}
```

Output:

Enter three numbers

10 15 20

20 is the largest number

Experiment No.
03

Aim: Study of if-else ladder.

Problem Statement: Write a program to find all the roots of a quadratic equation using if- else ladder.

Problem Definition:

Input: Coefficients of quadratic equation a, b and c

Processing: Using the formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

To check roots are

i) Real and equal [when b^2-4ac=0]

ii) Real and distinct [When b^2-4ac>0]

iii) Imaginary [when b^2-4ac<0]

Output: Type of roots

i) Real and equal

ii) Real and distinct

iii) Imaginary

Theory:

The if statement

In C programming language the else if ladder is a way of putting multiple ifs together when multipath decisions are involved. It is a one of the types of decision making and branching statements. A multipath decision is a chain of if's in which the statement associated with each else is an if. The general form of else if ladder is as follows -

```
if( condition 1)
{
        statement1;
}
```

```
else if(condition 2)
{
        statement 2;
}
else if( condition n)
{
        statement - n;
}
else
{
        default statement;
}
statement-x;
```

This construct is known as the else if ladder. The conditions are evaluated from the top of the ladder to downwards. As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder). When all the n conditions become false, then the final else containing the default statement will be executed.

Algorithm:

Step1: Start

Step2: read co-efficient of quadratic equation a, b, c

Step3: if 'a' is zero print "not a quadratic equation".

Calculate and print the answer using formula =-c/b

Else

Calculate d =

b^2-4ac If d = 0

Print "real and equal roots" Calculate roots using –b/2a Print both the roots

Else if d > 0

Printf "real and distinct roots"

Calculate roots using formula

(-b +sqrt(d)/2a) and (-b-sqrt(d)/2a)

Print both the roots.

Else Printf "imaginary roots"

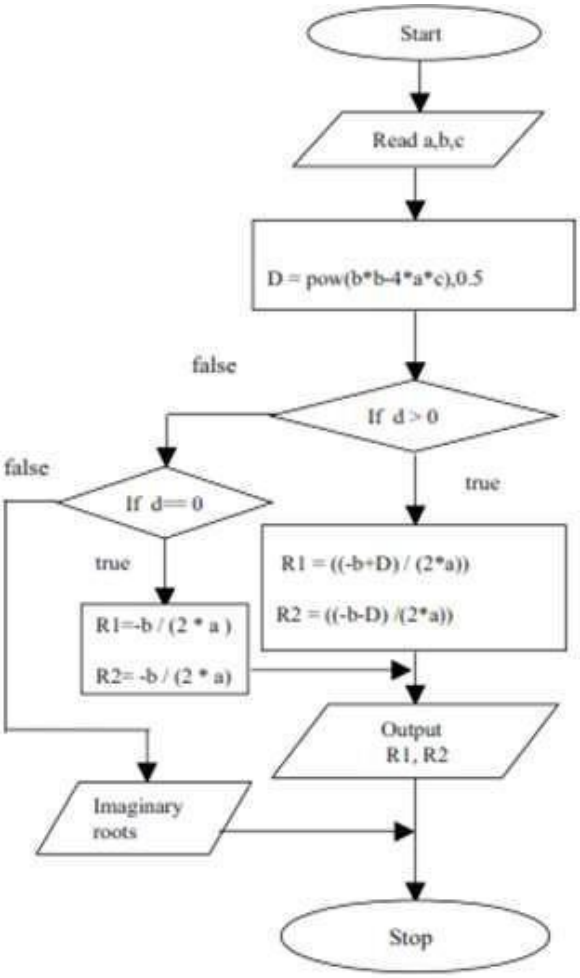
Calculate real parts as –b/2a

Calculate imaginary part as sqrt(-d)/2a

Print the roots using real and imaginary parts.

Step4: Stop

Flowchart:


Start
Read a,b,c
D = pow(b*b-4*a*c),0.5
false
If d > 0
true
false
If d== 0
true
R1 = ((-b+D) / (2*a))
R2 = ((-b-D) /(2*a))
R1=-b / (2 * a)
R2= -b / (2 * a)
Output
R1, R2
Imaginary
roots
Stop

Source Code:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
        float A , B , C , root1 , root2;
        float realp ,imgp , d;
        clrscr();
        printf(" Enter the values of A , B and
        C\n"); scanf("%f%f%f" , &A , &B , &C);

        if(A==0 || B==0 || C==0)
        {
                                        printf(" Error: Roots cannot be
                                        determined\n");
        }
        else
        {
                                        d = (B * B) - (4 * A * C);
                                        if(d<0)
                                        {
                printf(" Imaginary Roots\n");
                realp = -B/(2*A);
                imagp = sqrt(abs(d))/(2*A);
                printf(" Root1 = %f + i%f\n" , realp , imagp);
                printf(" Root2 = %f - i%f\n" , realp , imagp);
        }

        else if( d == 0)
        {
                printf(" Roots are real and
                equal\n"); root1 = -B/(2 * A);
```

```
root2 = Root1;
```

```
                printf(" Root1 = %f\n" , root1);
                printf(" Root2 =%f\n" , root2);
        }

        else if(d>0)
        {
                printf(" Roots are real and
                distinct\n"); root1 = (-B +
                sqrt(d))/(2*A);
                root2 = (-B - sqrt(d))/(2*A);
                printf(" Root1 = %f\n" , root1);
                printf(" Root2 = %f\n" , root2);
        }
    }
}
```

Output:

Output1:

```
Enter the values of A , B and
C 3 2 1
Imaginary roots
Root1 = -0.3333 + i0.471402
Root2 = -0.3333 - i0.471405
```

Output2:

```
Enter the values of A , B and
C 1 2 1
Roots are real and equal
Root1 = -1.0000
Root2 = -1.0000
```

Output3:

```
Enter the values of A , B and
C 3 5 2
Roots are real and distinct
Root1 = -0.666667
```

Experiment No. 04

Aim: Study of switch case.

Problem Statement: Write a program to implement an arithmetic calculator for addition, subtraction, multiplication, division and modulooperation using switch case.

Problem Definition:

Input: Accept two numbers and select operation Processing: Find the result of selected operation Output: Display result of operation

Theory:

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Syntax:

The syntax for a switch statement in C programming language is as follows:

```
switch(expression)
{
case constant-expression :
statement(s); break;/* optional */
case constant-expression :
statement(s); break;/* optional */

/* you can have any number of case statements
        */ default:/* Optional */
statement(s);
}
```

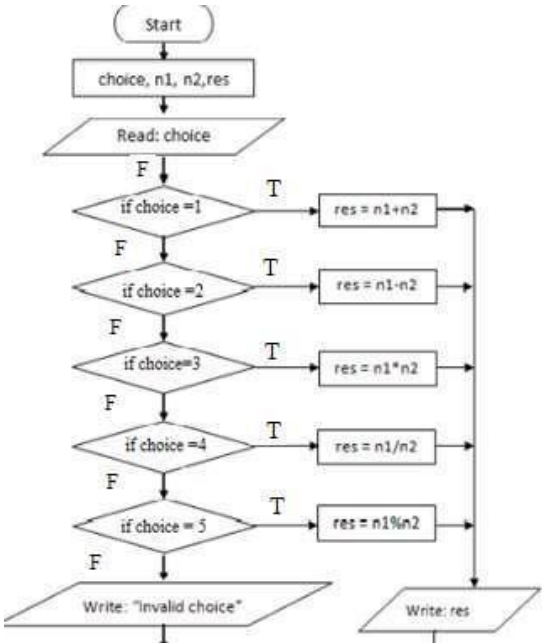
The following rules apply to a switch statement:

- The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Algorithm:

Step1: Start
Step2: Enter two numbers a, b
Step3: Enter a choice from 1. Add 2.Subtract 3.Multiply 4.Divide 5. Modulus
Step4: if choice = 1 then
res=a+b;
goto step 5
if choice = 2 then
res=a-b;
goto step
5
if choice = 3 then
res=a*b;
goto step
5
if choice = 4 then
res=a/b;
goto step
5
if choice = 5 then
res=a%b;
goto step
5
Step5: Display res
Step6: Stop

Flowchart:



Source Code:

```
# include<stdio.h>
void main()
{
        int ch, a, b,res;
        printf("enter two number\n
        :"); scanf("%d%d",&a,&b);
        printf("1. Add")
        printf("2.
        Subtraction");
        printf("3. Multiply");
        printf("4. Divide");
        printf("5. Modulus");
        printf("Enter your choice\n :");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:res=x+y;
                        break;
                case 2:
                res=x-y;
                        break;
                case 3:
                res=x*y;
                        break;
                case 4:
                res=x/y;
                        break;
                case 5:
                res=x%y;
                        break;
                default :printf("wrong choice");
```

```
        }
        printf("Result =%d\n",res);
}
```

Output:

enter two

number 2 3

1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus

1

Result =5

Experiment No: 05

Aim: Study of while loop.

Problem Statement: Write a program to check whether the given number is armstrong number or not using while loop.

Problem Definition:
Input: Accept a number
Processing: Compare the sum of cubes of each digit of a number with itself.
Output: Display whether entered number is Armstrong number or not.

Theory:
Awhile loop is a control flowstatement that allows code to be executed repeatedly based on a given boolean condition. The while loop can be thought of as a repetitionif statement.

The while construct consists of a block of code and a condition. The condition is evaluated, and if the condition is true, the code within the block is executed. This repeats until the condition becomes false. Because while loop checks the condition before the block is executed, the control structure is often also known as a pre-test loop. Compare with the do while loop, which tests the condition after the loop has executed.
An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself.
In other word “A number is Armstrong if it is equal the sum of cube of its digits.”
Example of Armstrong number is 371 because according to definition cube of its digits sum will be equal to number so
Armstrong number $371=(3)^3+(7)^3+(1)^3$

$371=27+343+1$

$371=371$

Algorithm:

Step 1: Start

Step 2: Read a number.

Step 3: org = num

Step 4: If num <=0 then goto step 9

Step 5: rem = num%10;

Step 6: sum = sum + rem*rem*rem;

Step 7: num = num/10;

Step 8: goto step 3

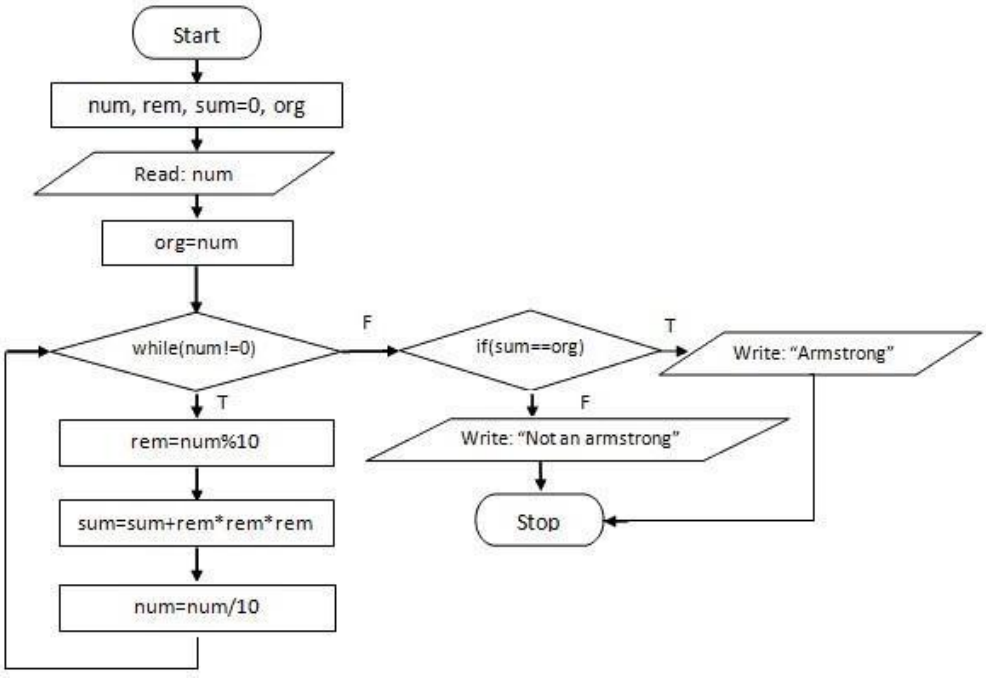
Step 9: if sum = org then

Print “Armstrong No”

Else Print “Not Armstrong No”

Step 10: Stop

Flowchart:



Source Code:

```
#include <stdio.h>
void main()
{
        int num, sum = 0, org,
        rem;    printf("Enter    a
        number\n");
        scanf("%d",&num);
        org = num;
        while( num != 0
        )
        {
                rem = num%10;
                sum = sum +
                rem*rem*rem; num =
                num/10;
        }
        if ( sum== org)
                printf("Armstrong number.");
        else
                printf("Not an Armstrong number.");
        return 0;
}
```

Output:Enter a number
371
Armstrong
number. Enter a
number 111
Not an Armstrong number.

Experiment 6

AIM:- Program for do-while loop

Problem Statement:- Write a program to find binary equivalent of a given decimal number using do-while loop.

Problem Definition:

Input: Accept a binary

Processing: Convert the binary number given into equivalent Decimal Number

Output: Equivalent decimal Number

Theory:- A while loop is a control flow statement that allows code to be executed repeatedly based on a given boolean condition. The while loop can be thought of as a repetitionif statement.

The while construct consists of a block of code and a condition. The condition is evaluated, and if the condition is true, the code within the block is executed. This repeats until the condition becomes false. Because while loop checks the condition before the block is executed, the control structure is often also known as a pre-test loop. Compare with the do while loop, which tests the condition after the loop has executed. A binary number is a number which is understandable by the computer. In other words " A binary number is a number consisting only of 1's & 0's"

Example:-A example of binary number is 100

And the decimal for the same is 4

Algorithm:-

Step1:-Take a binary number as the input.

Step2:-Divide the number by 10 and store the remainder into variable rem.

Step3:-decimal_num = decimal_num + rem * base;

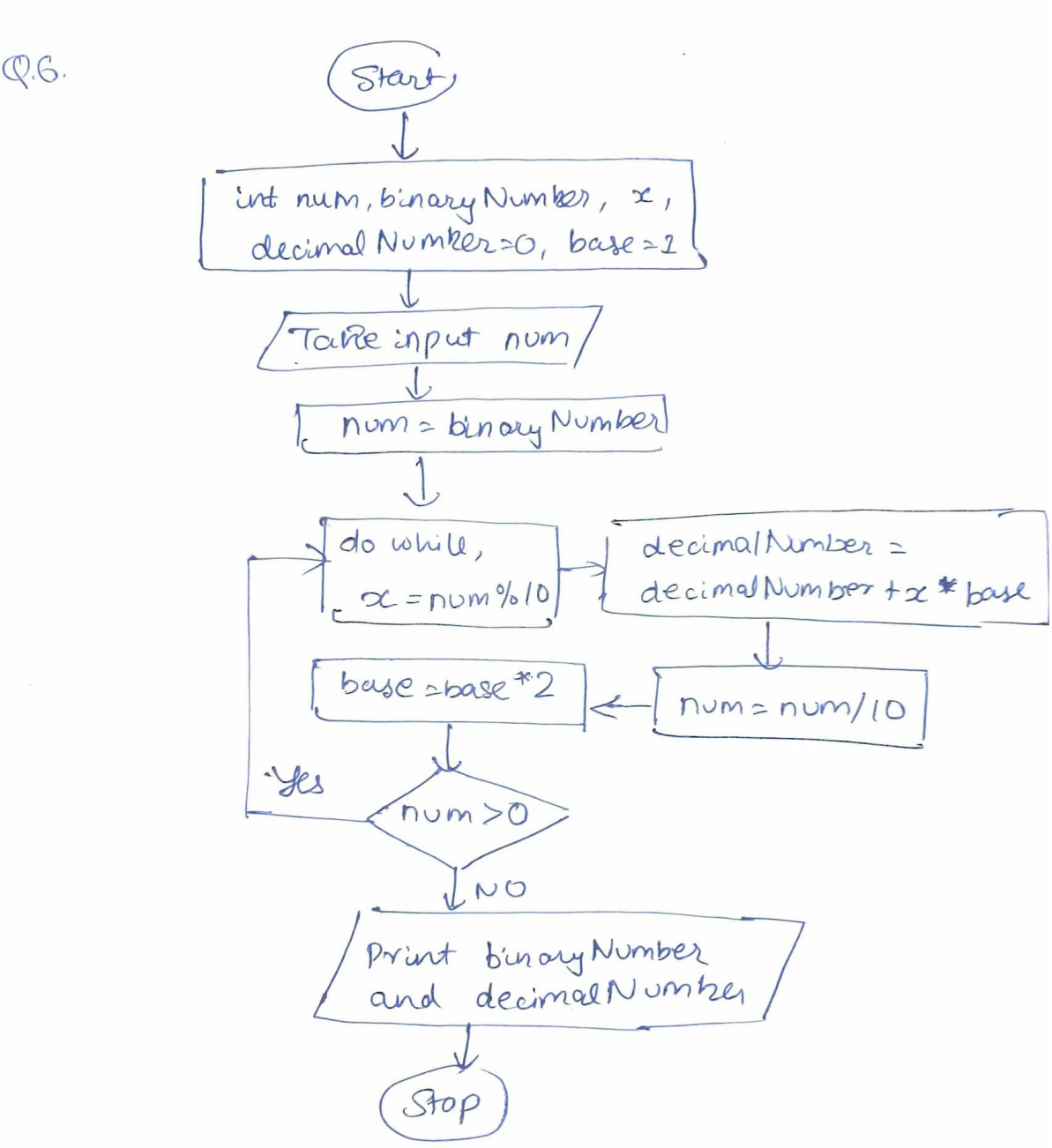
Step4:-Divide the quotient of the original number by 10.

Step5:-Multiply the base by 2.

Step6:-Print the decimal of the binary number.

Flowchart

Q.6.



Source Code:-

```c
#include <stdio.h>

int main()
{
  int num, binaryNumber, decimalNumber = 0, base = 1, x;

  printf ("Enter a binary number :\n");
  scanf (" %d", &num);

  binaryNumber = num;

  do
  {
      x = num % 10;
      decimalNumber = decimalNumber + x * base;
      num = num / 10;
      base = base * 2;
  } while (num > 0);

   printf ("The binary number is %d \n", binaryNumber);
   printf ("The decimal number is %d \n", decimalNumber);

   return 0;
}
```

Output:-

```
Enter a binary number :
100
The binary number is 100
The decimal number is 4
```

Experiment 7

Aim:- Program to implement for-loop

Problem Statement:- WAP to check whether an entered number is prime number or not using for-loop.

Problem Definition:

Input: Accept a number

Processing: Check whether the given number is prime or not

Output: Number is prime or not

Theory:

For loop is the loop in which the initialization statement is executed only once.

Then, the test expression is evaluated. If the test expression is FALSE, the loop is terminated.

However, if the test expression is true then the statements in the body of the loop are to be executed and the update expression is updated.

And again the test expression is evaluated.

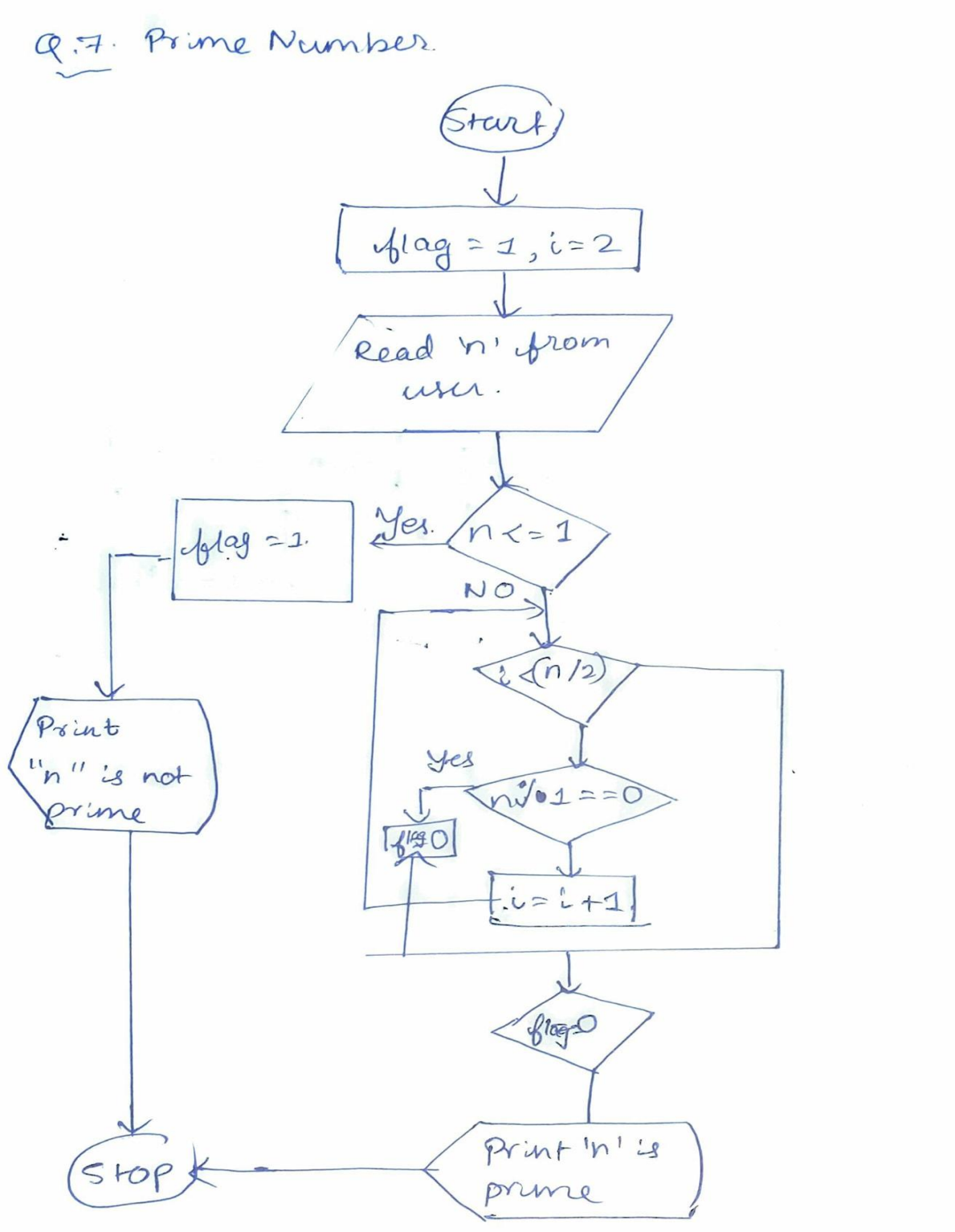
Prime Numbers is the numbers who are divided by 1 & the number itself .

Example:- An example of prime number is 2. As you can see 2 is the only number which is divisible by 1 and the number 2 itself

Algorithm:-

Step 1: Start
Step 2: Initialize variables num,flag=1, j=2
Step 3: Read num from user
Step 4: If num<=1
    Display "num is not a prime number"
    Goto step 7
Step 5: Repeat the steps until j<[(n/2)+1]
    If remainder of number divide j equals to 0,
        Set flag=0
    Goto step 6
    j=j+1
Step 6: If flag==0,
    Display num+" is not prime number"
    Else
    Display num+" n is prime number"
Step 7: Stop

Flowchart:


Q.7. Prime Number.
Start
flag = 1, i = 2
Read 'n' from user.
n <= 1
Yes.
flag = 1.
NO
i <= (n/2)
Yes
n%i == 0
flag = 0
i = i + 1
flag == 0
Print 'n' is prime
Print "n" is not prime
STOP

Source Code:-

```c
#include <math.h>
#include <stdio.h>

int main()
{
  int n, i, flag = 1;

  printf("Enter a number: ");
  scanf("%d", &n);

  for (i = 2; i <= sqrt(n); i++) {
     if (n % i == 0) {
        flag = 0;
        break;
     }
  }

  if (n <= 1)
  flag = 0;

  if (flag == 1)
     printf("%d is a prime number\n", n);
  else
     printf("%d is not a prime number\n", n);

  return 0;
}
```

Output:-

```
Enter a number: 2
2 is a prime number
```

Experiment 8

Aim: Program for nested for-loop

Problem Statement: WAP to generate the following given pattern using nested for-loop.

ABCDCBA
ABCBA
ABA
A

Problem Definition:

Input: No Input

Processing: Printing the given pattern using Nested For Loop

Output: The output will be given pattern

Theory:- The word NESTED means one thing inside the other thing.

You can write one type of loop in any other loop. In addition you can have any number of loop nested inside other. Below are some examples of nested loops.

In nested for loop the condition of the upper or the for loop which is written first is executed and then the for loop inside it is executed.

Example for nested for loop

```
for(initialization; condition; update)
{
 // statements

 for(initialization; condition; update)
 {
    // Inner loop statements
 }

 // statements
 }
```

Algorithm:-

Step1: Start

Step2: Enter the number.

Step3:

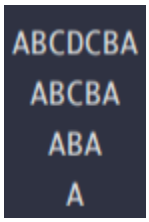
```
for i=1,i<=num,i++
        for j=num-i,j>=1,j--
        print “ ”
        for j=1,j<=i,j++
        print "j+64”
        for j=i-1,j>=1,j--
        print "j+64”
 print "\n"
```

Step4: Stop.

Source Code:-

```c
#include <stdio.h>
int main()
{
 int i, j, rows = 4;
 printf("\n\n");
 for(i = rows; i >= 1; i--)
 {
    for(j = 1; j <= rows - i; j++){
       printf(" ");
    }
    for(j = 1; j <= i; j++){
       printf("%c", j+64);
    }
    for(j = i - 1; j >= 1; j--){
       printf("%c",j+64);
    }
    printf("\n");
 }
 printf("\n\n");
}
```

Output :-



Experiment 9

Aim: Program for passing simple parameters to Function.

Problem Statement: WAP to find value of BIO using function, Where BIO is defined as

BIO=n!/(r!*(n-r)!), n and r are natural numbers.

Problem Definition:

Input: n, r

Processing: Formula n!/(r!*(n-r)!)

Output: ncr or BIO = n!/(r!*(n-r)!)

Theory:-

A for loop is a control flow of statement based on the Boolean condition in the syntax.

The loop can be thought of repition if statement.

In this for loop contain some code and then next for loop contain some code and before completing

previous loop it could not enter to the next loop

Here, we know that nCr = n!/(r!*(n-r)!)

Algorithm:

Step1: Start

Step2: Function int fact(int n)

Step3: int i, f = 1

Step4:

```
for (i = 1 ; i <= n; i++)
   {
      f = f * i;
   }
```

Step5: return the value of f

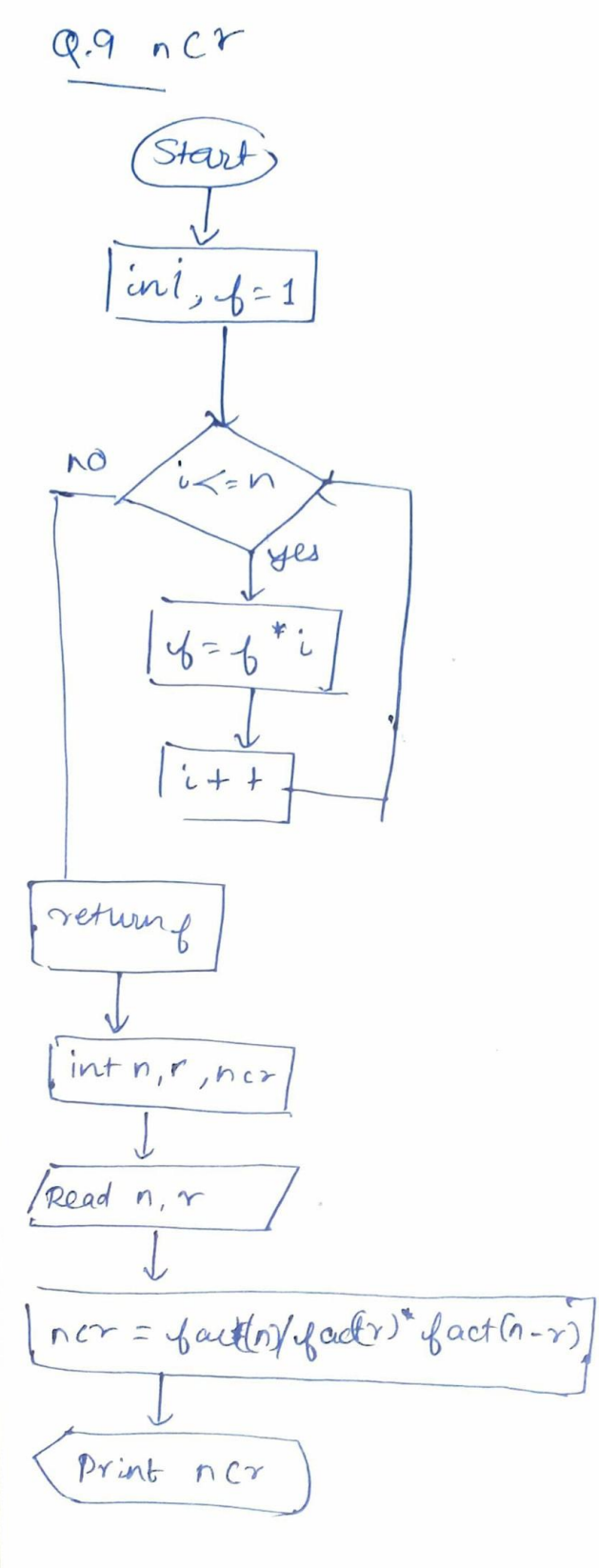
Step6: In the main function, enter the value of n and r

Step7: ncr = fact(n)/(fact(r)*fact(n-r));

Step8: Print the value of nCr or “BIO” or equal to n!/(r!*(n-r)!)

Step9: Stop

Flowchart:-


Q.9 nCr
Start
ini, f=1
NO
i<=n
yes
f=f*i
i++
return f
int n,r,ncr
Read n, r
ncr = fact(n)/fact(r)*fact(n-r)
Print nCr

Source Code:-

```c
#include <stdio.h>

// nCr = n!/r!*(n-r)!
int fact(int n)
{
  int i, f = 1;
  for(i = 1 ; i <= n; i++)
  {
      f = f * i;
  }
  return f;
}

int main()
{
  int n, r, ncr;

  printf("Enter a number n: ");
  scanf("%d",&n);
  printf("Enter a number r: ");
  scanf("%d",&r);
  ncr = fact(n)/(fact(r)*fact(n-r));
  printf("Value of BIO = n!/r!*(n-r)! = %d\n", ncr);
}
```

Output:-

```
Enter a number n: 5
Enter a number r: 4
Value of BIO = n!/r!*(n-r)! = 5
```

Experiment 10:

Aim: Study of recursive function

Problem Statement: WAP to find the GCD and LCM of two given numbers using recursive function.

Problem Definition:

Input: Two Numbers Given by User

Processing: Calculating the GCD & LCM of the numbers given by the users recursive function

Output: The GCD & LCM of the numbers given

Theory:

A function that calls itself is called recursive function. And, this technique is known as recursion.

The recursion is continued until a condition is met to prevent it.

To prevent infinite recursion, if……else(or similar approach) where one branch makes recursive call and the other doesn't.

Advantages and disadvantages of Recursive Function

Recursion makes program elegant. However, if performance is vital, use loops instead as recursion is usually much slower.

That being said, recursion is an important concept. It is frequently used in data structure and algorithms. For example, it is common to use recursion in problems such as tree traversal.

Example:-

```
void recurse()
{
        recurse();
}
int main()
{
         recurse();
}
```

Algorithm:-

Step1: Start

Step2: Function Prototype int gcd()

Step3: int a, b and If a is equal to b, return b

Step4: else return gcd(b % a, a)

Step5: Function Prototype int lcm()

Step6: return (a / gcd(a, b)) * b;

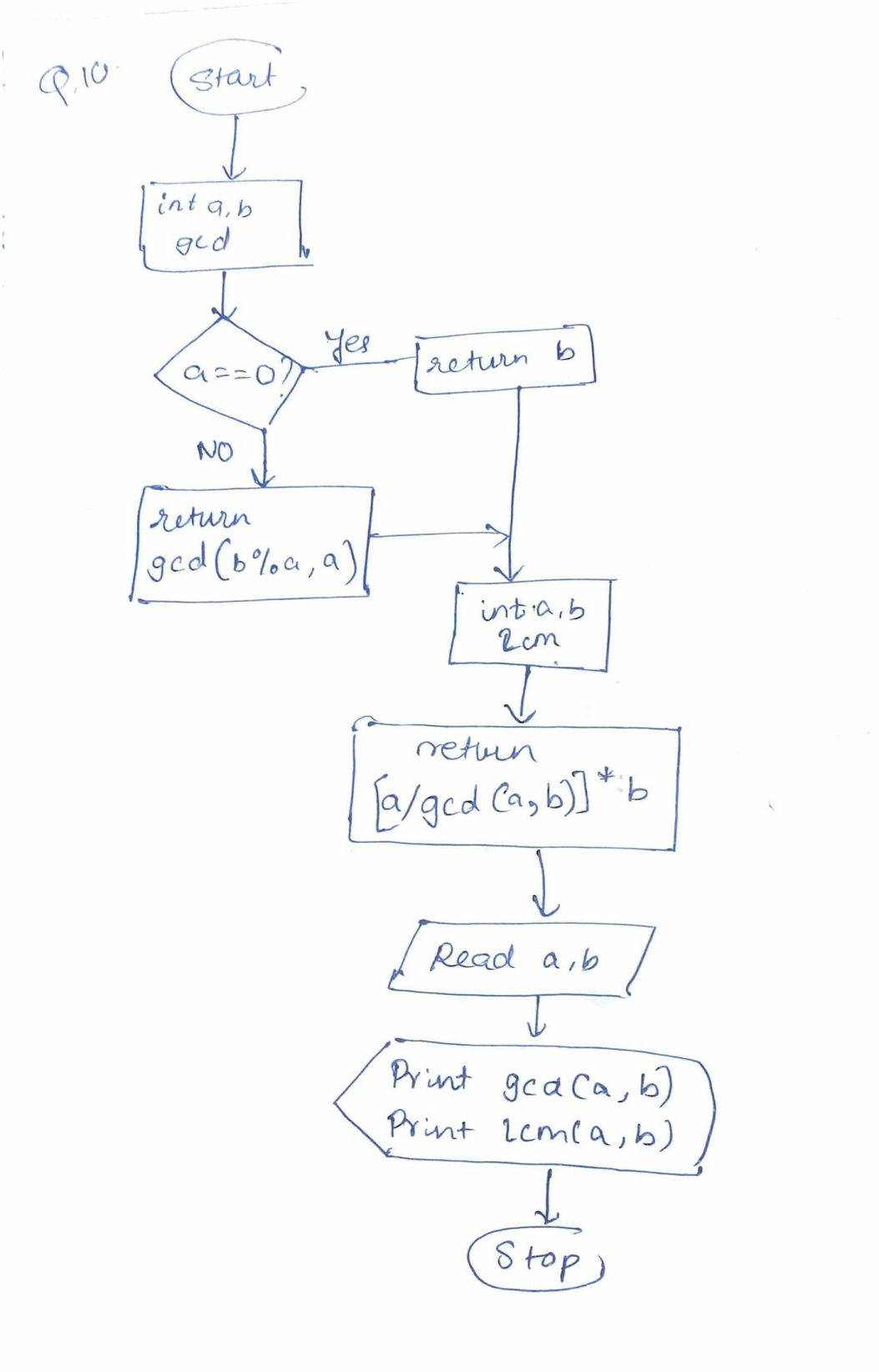
Step7: Main Function

Step8: Input a, b

Step9: Print “GCD” and “LCM”

Step10: Stop

Flowchart:-


Q.10
Start
int a,b
gcd
a==0?
Yes
return b
NO
return
gcd(b%a,a)
int a,b
lcm
return
[a/gcd(a,b)] * b
Read a,b
Print gcd(a,b)
Print lcm(a,b)
Stop

Source Code:-

```c
#include <stdio.h>
int gcd(int a, int b)

{
  if (a == 0)
     return b;
  else
     return gcd(b % a, a);
}

int lcm(int a, int b)
{
  return (a / gcd(a, b)) * b;
}

int main()
{
  int a, b;
  printf("Enter first number: ");
  scanf("%d", &a);

  printf("Enter second number: ");
  scanf("%d", &b);

  printf("GCD of %d and %d = %d \n", a, b, gcd(a, b));
  printf("LCM of %d and %d = %d \n", a, b, lcm(a, b));

  return 0;
}
```

Output:

```
Enter first number: 5
Enter second number: 10
GCD of 5 and 10 = 5
LCM of 5 and 10 = 10
```

Name: Atharva Auti
FE 15
Roll no. 01