NAME: Yash Sarang
ROLL NO.: 47
CLASS: D1AD

CP LAB ASSIGNMENT 2

THEORY:
A statement that is used to control the flow of execution in a program is called control structure. It combines instruction into logical units. Logical unit has one entry point and one exit point.

Types of control structures
1. Sequence
2. Selection
3. Repetition
4. Function call

1. SEQUENCE: Statements are executed in a specified order. No statement is skipped and no statement is executed more than once.
2. SELECTION: It selects a statement to execute on the basis of condition. Statement is executed when the condition is true and ignored when it is false e.g if, if else, switch structures.
   Selection structures allow the computer to make decisions in your programs. It selects a statement or set of statements to execute on the basis of a condition.

   Types:
   - If-else structure:
     The if-else statement is an extension of the simple if statement. It executes one statement if it is true and another one if the condition is false.
     1. If statement:
        The simplest if structure involves a single executable statement. Execution of the statement occurs only if the condition is true.
        Syntax:
        if (condition)
        statement;
     2. If-else statement:
        In an if-else statement if the condition is true, then the true statement(s), immediately following the if-statement are executed otherwise the false statement(s) are executed. The use of else

basically allows an alternative set of statements to be executed if the condition is false.

Syntax:

```
If (condition)
{
Statement(s);
}
else
{
statement(s);
}
```

3. If- else if statement:
   It can be used to choose one block of statements from many blocks of statements. The condition which is true is that only its block of statements is executed and remaining are skipped.
   Syntax:

```
if (condition)
{
statement(s);
}
else if (condition)
{
statement(s);
}
else
{
(statement);
}
```

4. Nested If:
   In a nested-if statement if the first if condition is true the control will enter inner if. If this is true the statement will execute otherwise control will come out of the inner if and the else statement will be executed.
   Syntax:

```
If (condition)
        if(condition)
{
statement(s);
}
else
```

{
statement(s);
}
else
{
statement(s);
}

- Switch structure:
  It is used when there are many choices and only one statement is to be executed.Switch statement is alternative to nested if-else.it is executed when there are many choices and only one is to be executed.
  Syntax:
  switch(expression)
  {
  case 1:
  statement;
  break;
  case 2:
  statement;
  break;
  .
  .
  .
  .
  case N:
  statement;
  break;
  default:
  statement;
  }
- Conditional Operator:
  It is also a decision-making statement. It is an alternative to if-else.it is known as ternary operator as it has three operands.
  Syntax:
  (condition)? true-case statement: false case statement;

3. REPETITION: In this structure the statements are executed more than one time. It is also known as iteration or loop e.g while loop, for loop do-while loops etc.
   Purpose:
   - Execute statements for a specified number of times.
   - To use a sequence of values.

Types:

- While Loop:
  It executes one or more statements while the given condition remains true.It is useful when the number of iterations is unknown.
  Syntax:
  initialization
  while (condition)
  {
  statement;
  increment/decrement;
  }
- Do- while Loop:
  Do while loops are useful where the loop is to be executed at least once. In do while loop condition comes after the body of loop.This loop executes one or more statements while the given condition is true.
  Syntax:
  initialization
  do
  {
  statement(s);
  increment/decrement;
  }
  while (condition);
- For Loop:
  For loops are used when the number of iterations is known before entering the loop. It is also known as a counter-controlled loop.
  Syntax:
  for (initialization;condition;increment/decrement)
  {
  statement()s;
  }

Nested Loop:
A loop within a loop is called a nested loop. In this the outer loop is used for counting rows and the internal loop is used for counting columns.Any loop can be used as the inner loop of another loop.
Syntax:

```
for (initialization;condition;increment/decrement)
{
  for(initialization;condition,increment/decrement)
  {
```

```
    statements(s);
    }
}
```

4. FUNCTION CALL: It is used to invite or call a piece of code or statement. In this case control jumps from the main program to that piece of code and then returns back to the main program.
    - Continue statement:
      Continue statement transfer the control to the start of the block.It is used in the body of the loop.
    - Break statement:
      It is a tool to take out the control.It transfers control to the end of block.It is used in the body of loop and switch statements.
    - Goto statement:
      It is an unconditional transfer of control.It transfers the control to the specific point.The goto statement is marked by a label statement.Label statement can be used anywhere in the function above or below the goto statement.
      It is written as goto label;
      where label is an identifier
      label: statement

SECTION 1:

(**a**)

AIM: To find roots of a quadratic equation.

ALGORITHM:

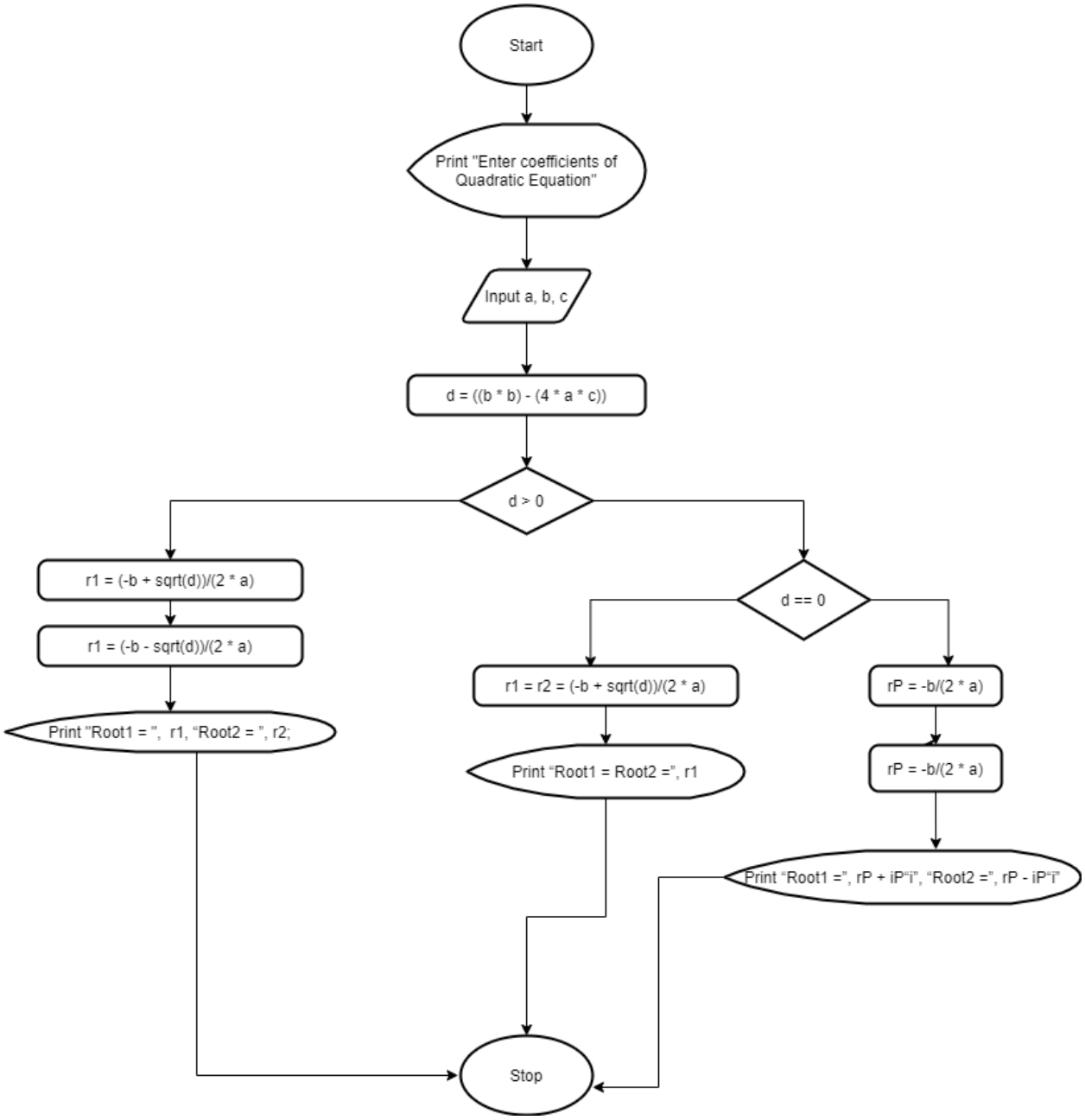
1. Start
2. Print “Enter coefficients of Quadratic Equation: ”
3. Input a, b, c;
4. d = ((b * b) - (4 * a * c));
5. if (d > 0)

```
{
        r1 = (-b + sqrt(d))/(2 * a);
        r2 = (-b - sqrt(d))/(2 * a);
        Print “Root1 = ”, r1, “Root2 = ”, r2;

}
else if (d = 0)
{
        r1 = r2 = -b/(2 * a);
        Print “Root1 = Root2 =”, r1;
}
else
{
        rP = -b/(2 * a);
        iP = sqrt(-d)/(2 * a);
        Print “Root1 =”, rP + iP“i”, “Root2 =”, rP - iP“i”;
}
```

6. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
#include<math.h>
int main() {
    double a, b, c, d, r1, r2, rP, iP;
    printf("\n Enter coefficients of Quadratic Equation: ");
    scanf("%lf %lf %lf", &a, &b, &c);
    d = ((b * b) - (4 * a * c));
    if (d > 0)
```

```
    {
        r1 = (-b + sqrt(d))/(2 * a);
        r2 = (-b - sqrt(d))/(2 * a);
        printf("\n Root1 = %.2lf \n Root2 = %.2lf", r1, r2);
    }
    else if (d == 0)
    {
        r1 = r2 = -b/(2 * a);
        printf("\n Root1 = Root2 = %.2lf", r1);
    }
    else
    {
        rP = -b/(2 * a);
            iP = sqrt(-d)/(2 * a);
            printf("\n Root1 = %.2lf + %.2lfi \n Root2 = %.2lf - %.2lfi, rP, iP, rP, iP");
    }
    return 0;
}
```

OUTPUT:

```
Enter coefficients of Quadratic Equation: 1 -8 12

 Root1 = 6.00
 Root2 = 2.00
```

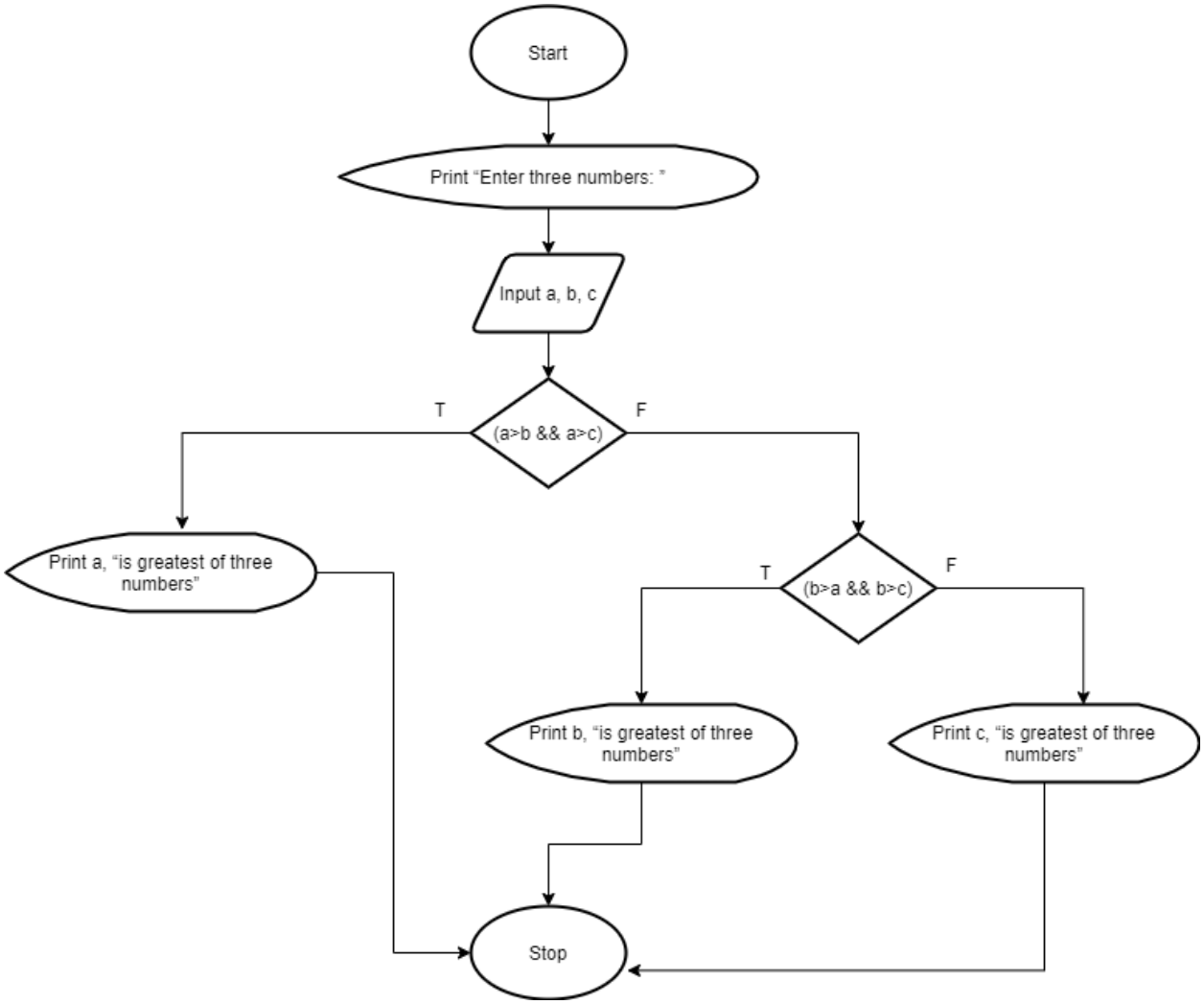
(**b**)

AIM: To find the greatest of three numbers.

ALGORITHM:

1. Start
2. Print “Enter three numbers: ”;
3. Input a, b, c;
4. if (a>b && a>c)
   Print a, “is greatest of three numbers”;
   else if (b>a && b>c)
   Print b, “is greatest of three numbers”;
   else
   Print c, “is greatest of three numbers”;
5. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
   int a, b, c;
   printf("Enter three numbers: ");
   scanf("%d %d %d", &a, &b, &c);
   if (a>b && a>c)
            printf("%d is greatest of three numbers.", a);
   else if (b>a && b>c)
            printf("%d is greatest of three numbers.", b);
   else
            printf("%d is greatest of three numbers.", c);
   return 0;
}
```

OUTPUT:

```
Enter three numbers: 2 4 7
7 is greatest of three numbers.
```

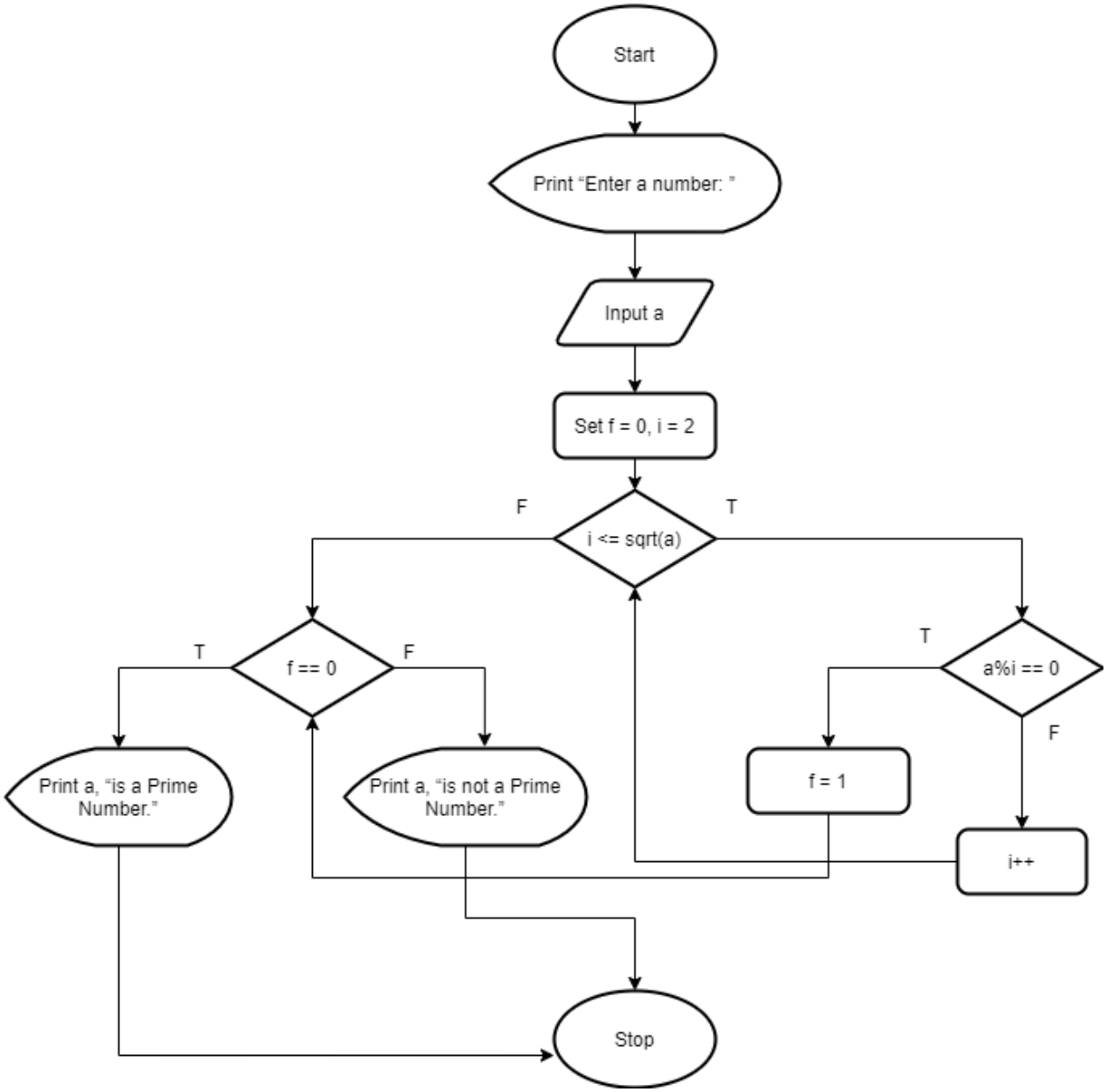
(**c**)

AIM: To test whether the given number is Prime or not a Prime.

ALGORITHM:

1. Start
2. Print “Enter a number: ”;
3. Input a;
4. Set f = 0, i = 2;
5. Repeat until (i <= sqrt(a))
   {
   if (a%i == 0)
   {
   f = 1;
   goto step 6;
   }
   i++;
   }
6. if (f == 0)
   Print a, “is a Prime Number.”
   else
   Print a, “is not a Prime Number.”
7. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
#include<math.h>
int main() {
    int a;
    printf("Enter a number: ");
    scanf("%d", &a);
    int f = 0, i = 2;
    while(i <= sqrt(a))
    {
```

```
        if (a%i == 0)
                {
                        f = 1;
                        break;
        }
            i++;
    }
    if (f == 0)
            printf("%d is a Prime Number.", a);
    else
       printf("%d is not a Prime Number.", a);
    return 0;
}
```

OUTPUT:

```
Enter a number: 4
4 is not a Prime Number.
```

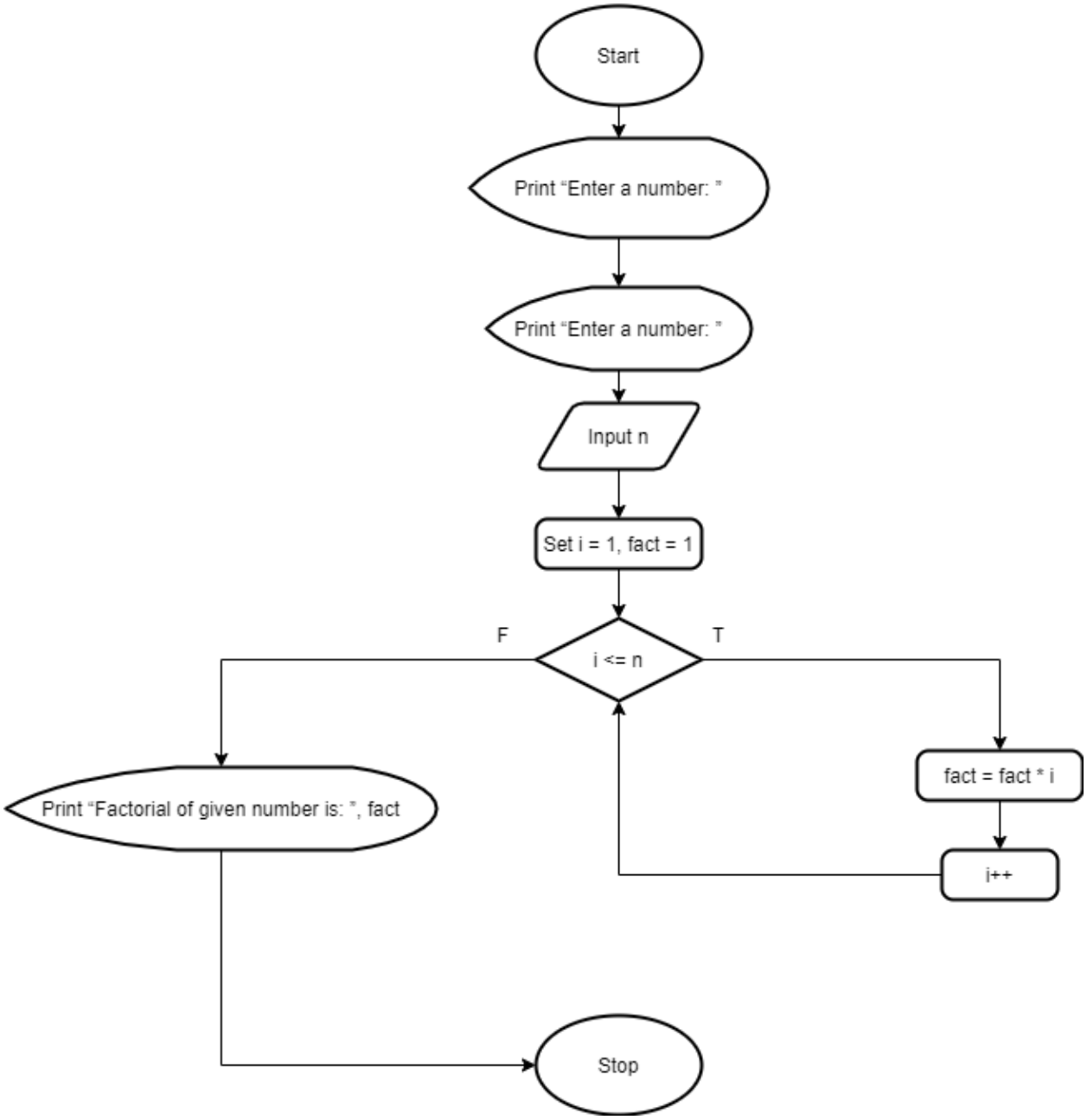
(**e**)

AIM: To find the Factorial of a number.

ALGORITHM:

1. Start
2. Print “Enter a number: ”
3. Input n;
4. Set i = 1, fact = 1;
5. Repeat until(i <= n)
   {
   fact = fact * i;
   i++;
   }
6. Print “Factorial of given number is: ”, fact;
7. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    int i = 1, fact = 1;
    while (i <= n)
    {
```

```
        fact = fact * i;
            i++;
    }
    printf("Factorial of given number is: %d", fact);
    return 0;
}
```

OUTPUT:

```
Enter a number: 5
Factorial of given number is: 120
```

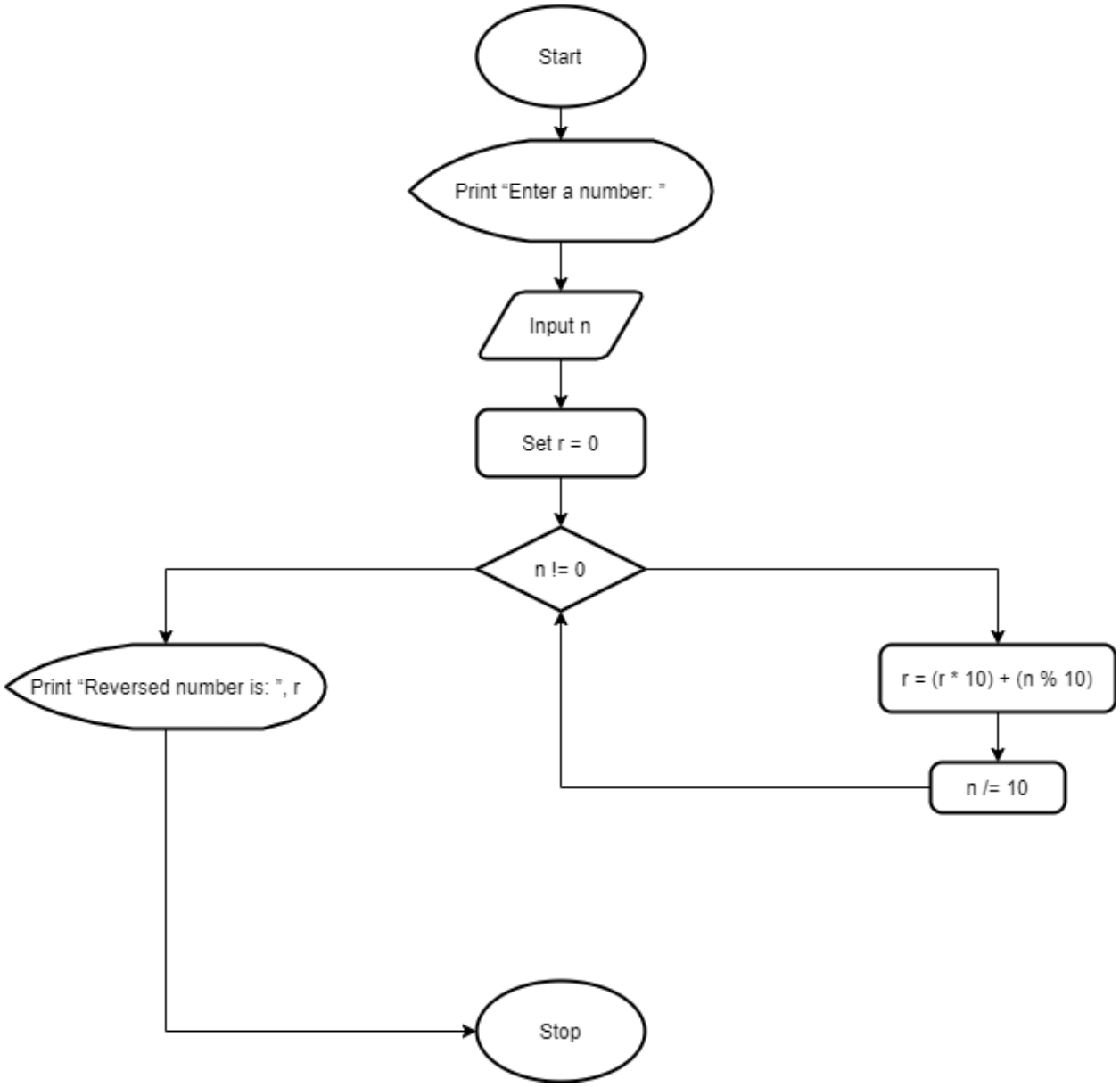
(**f**)

AIM: To reverse given number.

ALGORITHM:

1. Start
2. Print “Enter any number: ”;
3. Input n;
4. Set r = 0;
5. Repeat until (n != 0)
   {
   r = (r * 10) + (n % 10);
   n /= 10;
   }
6. Print “Reversed number is: ”, r;
7. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
    int n, r = 0;
    printf("Enter any number: ");
    scanf("%d", &n);
    while (n != 0)
    {
        r = (r * 10) + (n % 10);
          n /= 10;
```

```
    }
    printf("Reversed number is: %d", r);
    return 0;
}
```

OUTPUT:

```
Enter any number: 2345
Reversed number is: 5432
```

SECTION 2:

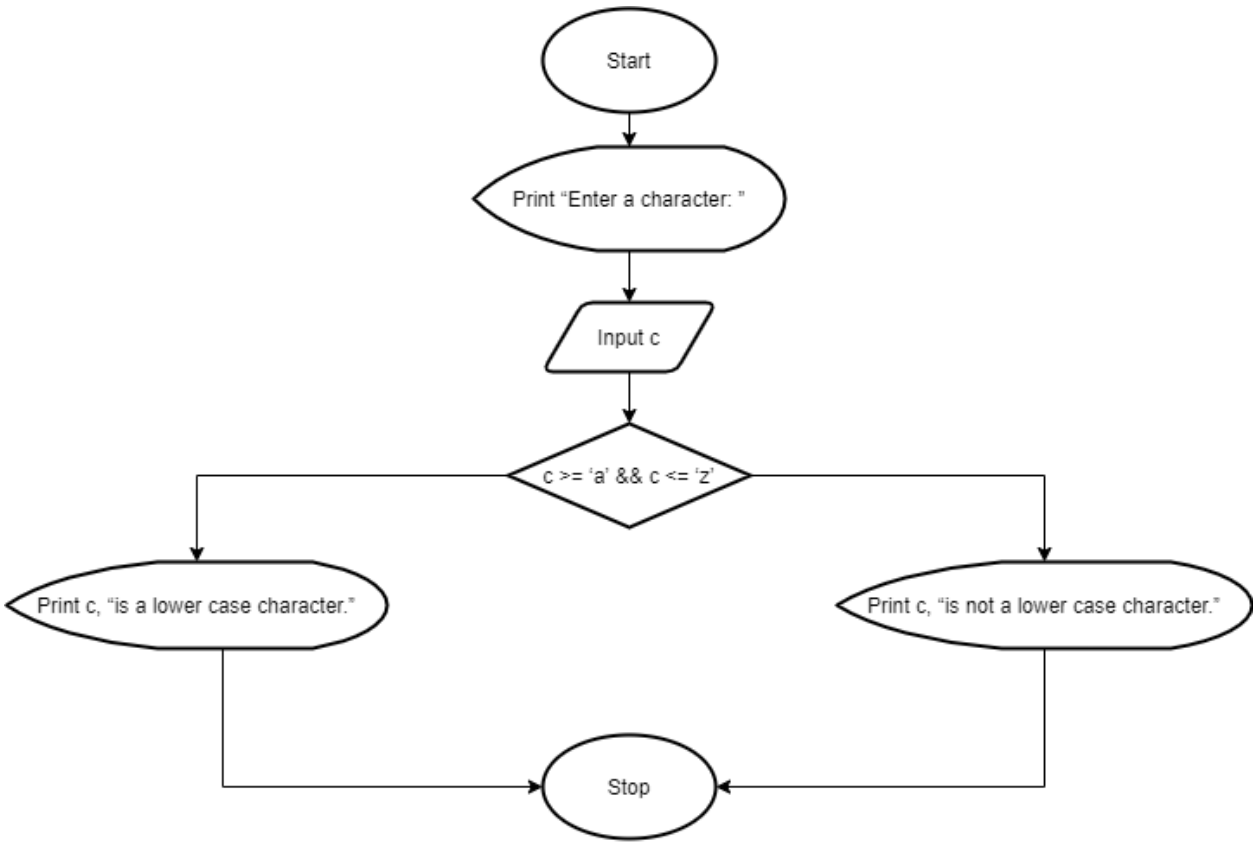
(**a**)

AIM: To check whether the character entered by keyboard is lowercase or not.

ALGORITHM:

1. Start
2. Print “Enter a character: ”;
3. Input c;
4. (c >= ‘a’ && c <= ‘z’)?(Print c, “is a lower case character.”):(Print c, “is not a lower case character.”);
5. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
   char c;
   printf("Enter a character: ");
   scanf("%c", &c);
```

```
    (c >= 'a' && c <= 'z')?(printf("%c is a lower case character.", c)):(printf("%c is not a
lower case character.", c));
    return 0;
}
```

OUTPUT:

```
Enter a character: R
R is not a lower case character.
```

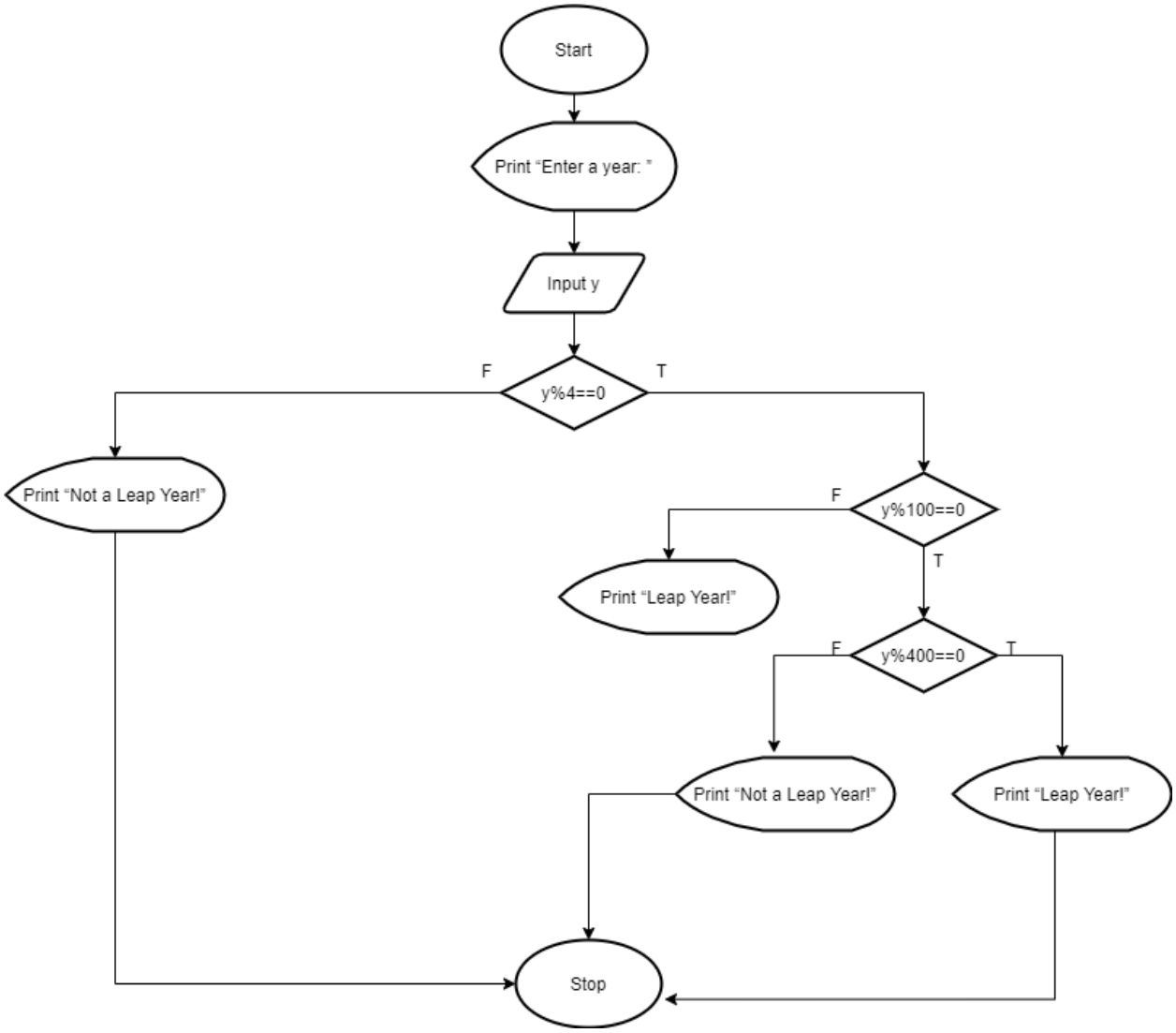
(**b**)

AIM: To check whether the entered year is a leap year or not.

ALGORITHM:

1. Start
2. Print “Enter a year: ”;
3. Input y;
4. (y%4==0)?((y%100==0)?((y%400==0)?(Print “Leap Year!”):(Print “Not a Leap Year!”)):(Print “Leap Year!”)):(Print “Not a Leap Year!”);
5. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
    int y;
    printf("Enter a year: ");
    scanf("%d", &y);
    (y%4==0)?((y%100==0)?((y%400==0)?(printf("Leap Year!")):(printf("Not a Leap
Year!"))):(printf("Leap Year!"))):(printf("Not a Leap Year!"));
    return 0;
}
```

OUTPUT:

```
Enter a year: 2020
Leap Year!
```

SECTION 3:

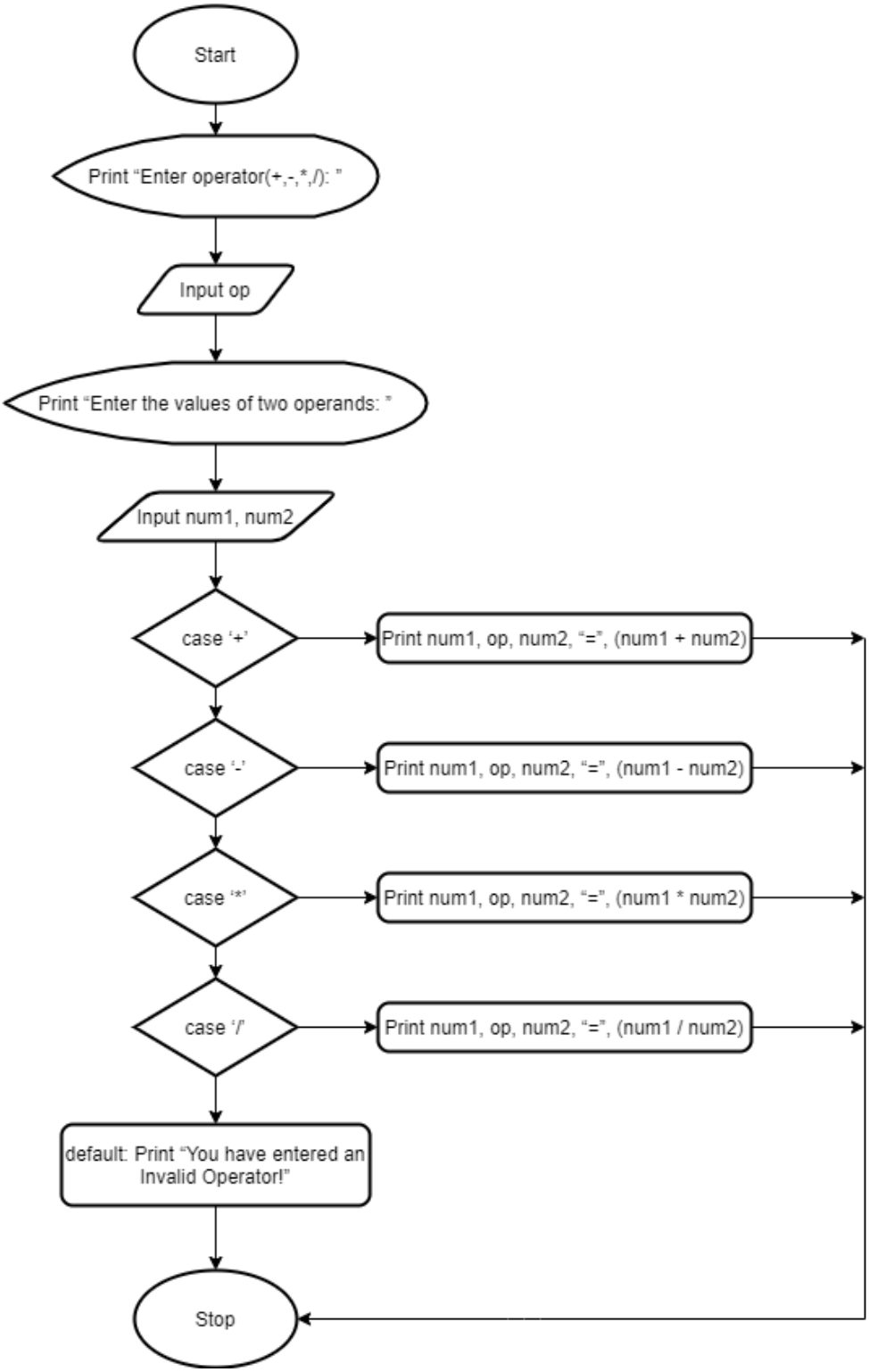
(**a**)

AIM: To design a simple interactive calculator.

ALGORITHM:

1. Start
2. Print “Enter operator(+,-,*,/): ”;
3. Input op;
4. Print “Enter the values of two operands: ”;
5. Input num1, num2;
6. switch(op)
   {
   case ‘+’: Print num1, op, num2, “=”, (num1 + num2);
   break;
   case ‘-’: Print num1, op, num2, “=”, (num1 -num2);
   break;
   case ‘*’: Print num1, op, num2, “=”, (num1 * num2);
   break;
   case ‘/’: Print num1, op, num3, “=”, (num1 / num2);
   break;
   default: Print “You have entered an Invalid Operator!”;
   }
7. Stop

FLOWCHART:



PROGRAM C CODE:

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

```
    char op;
        float num1, num2;
        printf("\n Please Enter an Operator (+, -, *, /)  :  ");
        scanf("%c", &op);
        printf("\n Please Enter the Values for two Operands: ");
        scanf("%f%f", &num1, &num2);
        switch(op)
        {
                case '+':
                        printf("\n %.2f + %.2f  = %.2f", num1, num2, num1 + num2);
                        break;
                case '-':
                        printf("\n %.2f - %.2f  = %.2f", num1, num2, num1 - num2);
                        break;
                case '*':
                        printf("\n %.2f * %.2f  = %.2f", num1, num2, num1 * num2);
                        break;
                case '/':
                        printf("\n %.2f / %.2f  = %.2f", num1, num2, num1 / num2);
                        break;
                default:
                        printf("\n You have entered an Invalid Operator!");

        }
    return 0;
}
```

OUTPUT:

```
Please Enter an Operator (+, -, *, /)  :  +

Please Enter the Values for two Operands: 3 7

3.00 + 7.00  = 10.00
```

(**b**)

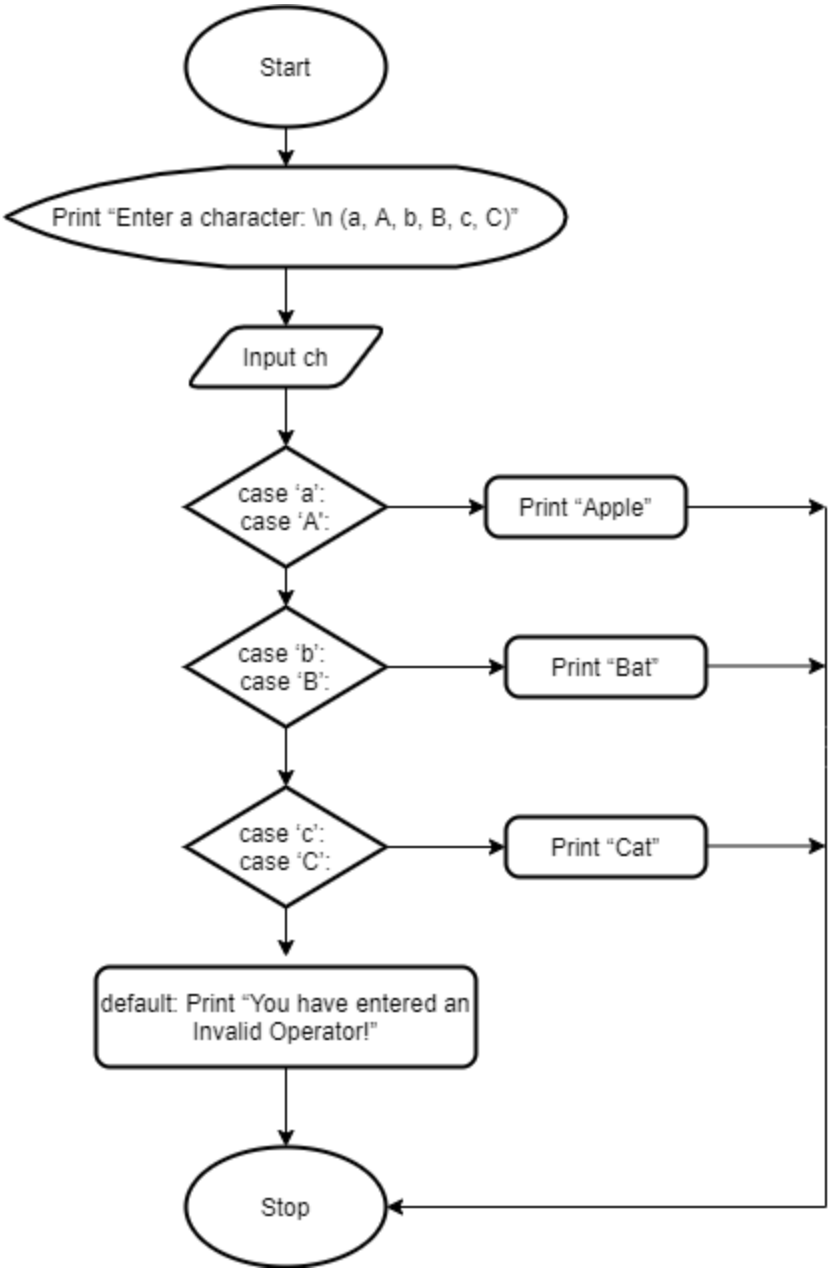
AIM: To scan character from keyboard and display ‘Apple’ if character is ‘a’ or ‘A’, ‘Bat’ if character is ‘b’ or ‘B’, ‘Cat’ if the character is ‘c’ or ‘C’.

ALGORITHM:

1. Start
2. Print “Enter a character: \n (a, A, b, B, c, C)”;

3. Input ch;
4. switch(ch)
   {
   case ‘a’: case ‘A’: Print “Apple”;
   case ‘b’: case ‘B’: Print “Bat”;
   case ‘c’: case ‘C’: Print “Cat”;
   default : Print “Invalid Character!”;
   }
5. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
    char ch;
    printf("Enter a character: \n(a, A, b, B, c, C): \n");
    scanf("%c", &ch);
    switch(ch)
    {
        case 'a': case 'A': printf("Apple"); break;
        case 'b': case 'B': printf("Bat"); break;
        case 'c': case 'C': printf("Cat"); break;
        default: printf("Invalid Character!"); break;
    }
    return 0;
}
```

OUTPUT:

SECTION 4:

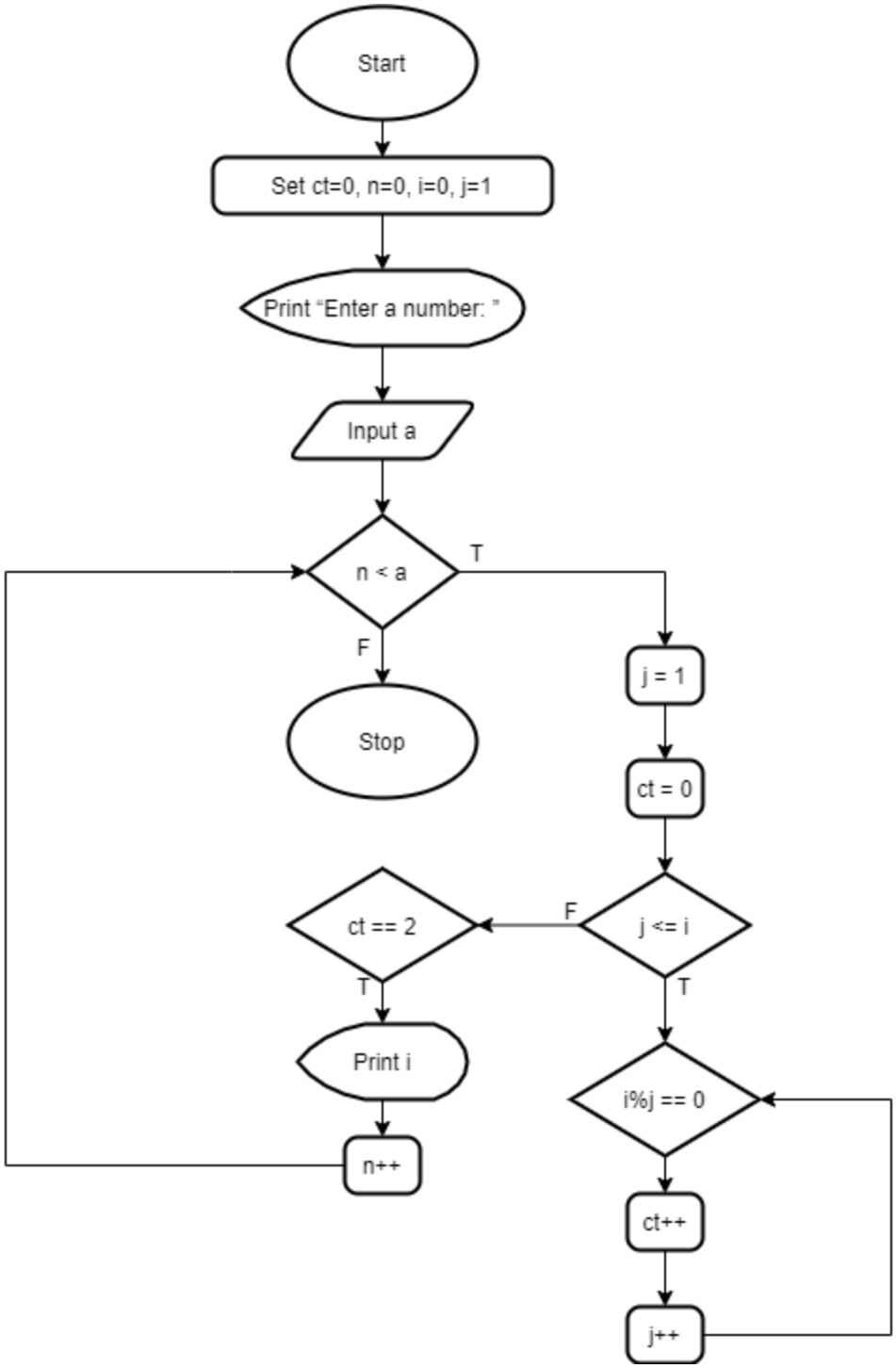
(**a**)

AIM: To print first n prime numbers.

ALGORITHM:

1. Start
2. Set ct=0, n=0, i=0, j=1;
3. Print “Enter a number: ”;
4. Input a;
5. Repeat until (n < a)
   {
   j=1
   ct=0;
   Repeat until (j<=i)
   {
   if(i%j==0)
   ct++;
   j++;
   }
   if(ct==2)
   {
   Print i;
   n++;
   }
   }
6. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
    int ct=0,n=0,i=1,j=1, a;
```

```
    printf("Enter a number: ");
    scanf("%d", &a);
    while(n<a)
    {
        j=1;
        ct=0;
        while(j<=i)
        {
            if(i%j==0)
            ct++;
            j++;
        }
        if(ct==2)
        {
            printf("\n%d ",i);
            n++;
        }
        i++;
}
    return 0;
}
```

OUTPUT:

```
Enter a number: 8
2
3
5
7
11
13
17
19
```

(**b**)

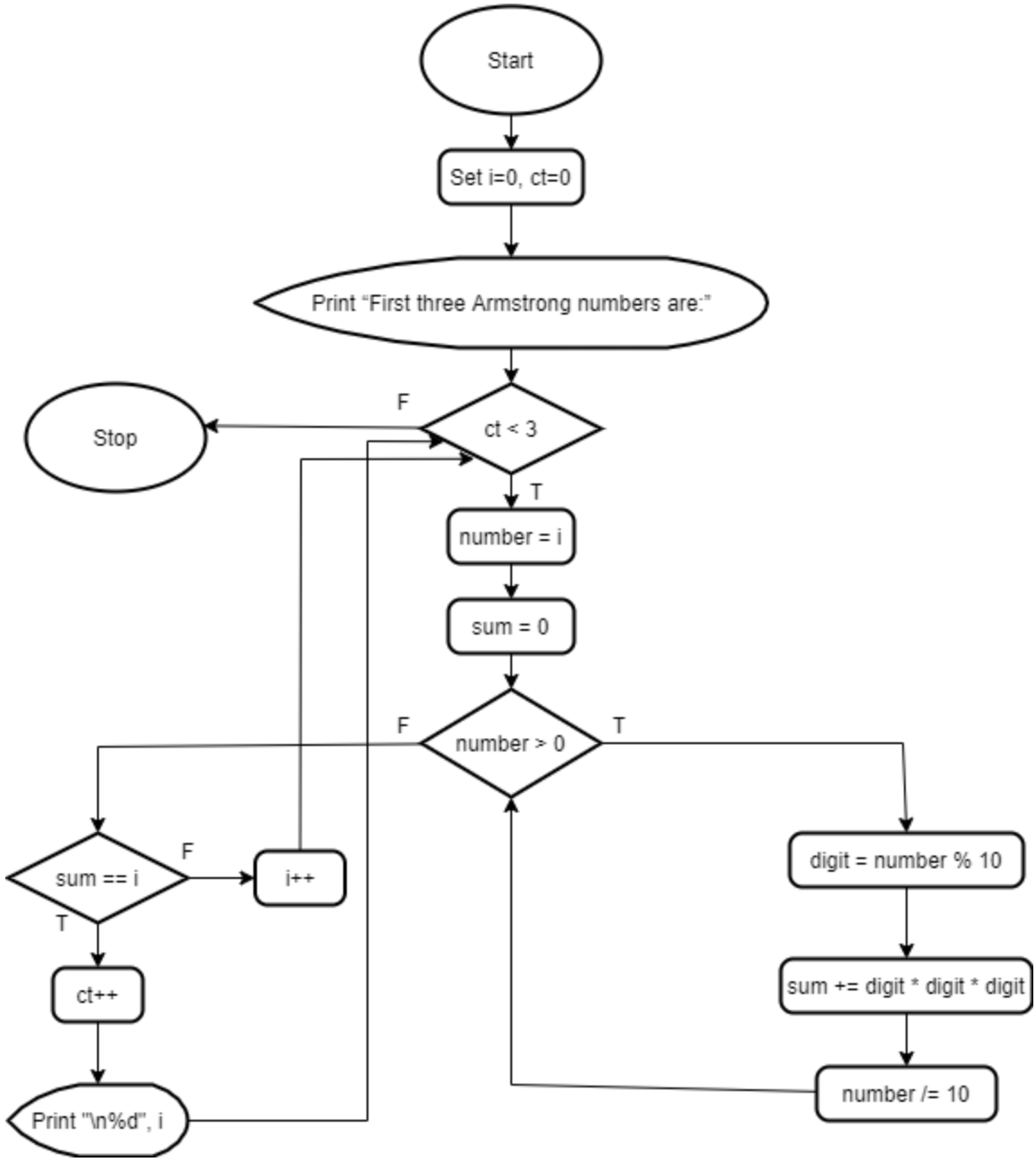
AIM: To print the first three Amstrong numbers.

ALGORITHM:

1. Start
2. Set i=0, ct=0;
3. Print “First three Armstrong numbers are:”;
4. Repeat until (ct < 3)

5. {
```
number = i;
sum = 0;
Repeat until (number > 0)
{
    digit = number % 10;
    sum += digit * digit * digit;
    number /= 10;
}
if (sum == i)
{
    ct++;
    Print "\n%d", i;
}
i++;
```
}
6. Stop

FLOWCHART:



PROGRAM C CODE:

```
#include <stdio.h>
int main() {
    int number, i=0, digit, sum, ct=0;
    printf("First three Armstrong numbers are: ");
    while (ct < 3)
    {
        number = i;
        sum = 0;
```

```
        while (number > 0)
        {
            digit = number % 10;
            sum += digit * digit * digit;
            number /= 10;
        }
        if (sum == i)
        {
            ct++;
            printf("\n%d", i);
        }
        i++;
    }
    return 0;
}
```

OUTPUT:

```
First three amstrong numbers are:
0
1
153
```

(**c**)

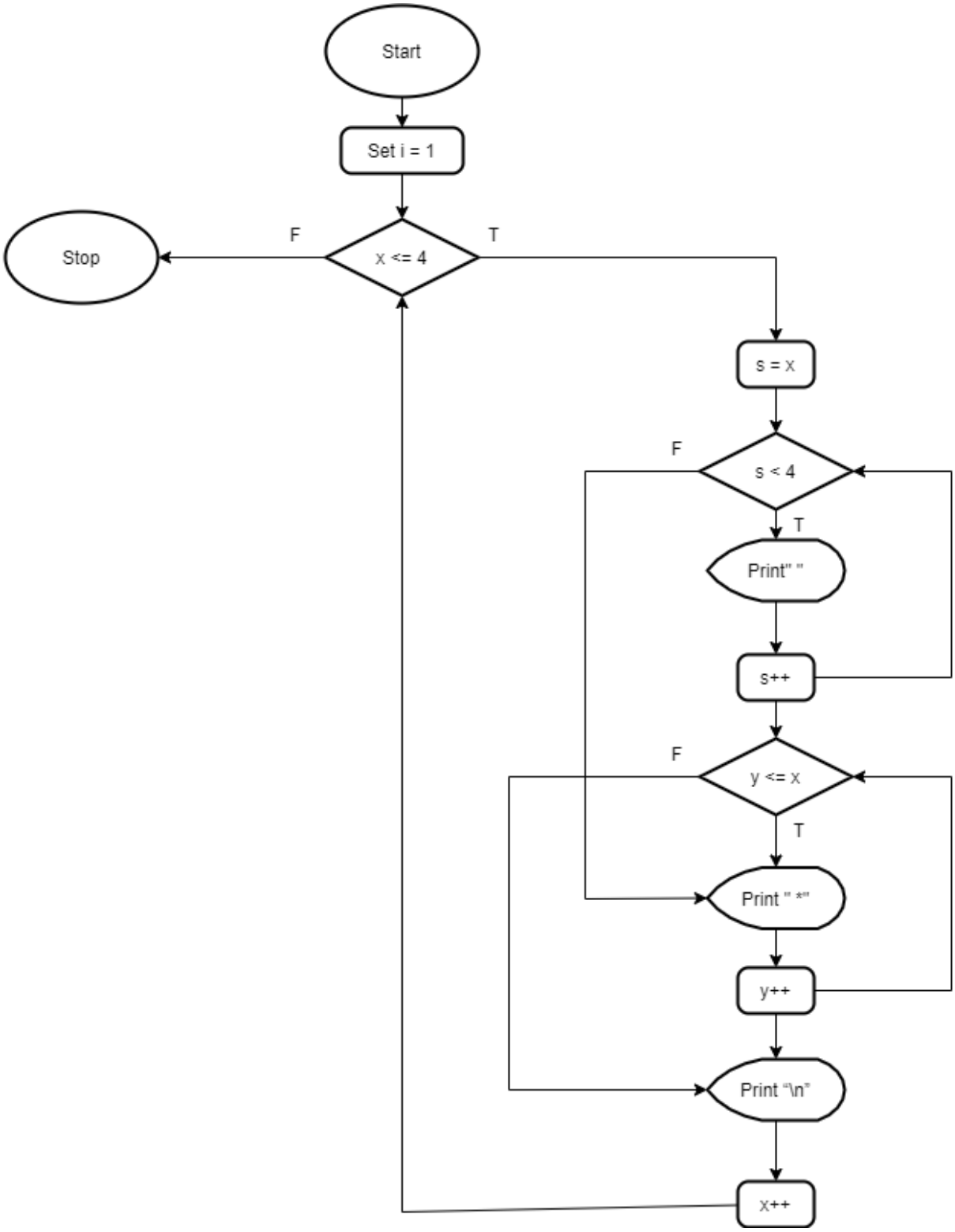
AIM: To print following pattern

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

ALGORITHM:

1. Start
2. for(x = 1; x <= 4; x++)
   {
   for(s = x; s < 4; s++)
   Print" ";
   for(y = 1; y <= x; y++)
   Print " *";
   Print “\n”;
   }
3. Stop

FLOWCHART:



PROGRAM C CODE:

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

```
int main() {
    int s, x, y;
    for(x = 1; x <= 4; x++)
    {
        for(s = x; s < 4; s++)
            printf(" ");
        for(y = 1; y <= x; y++)
            printf(" *");
    printf("\n");
    }
    return 0;
}
```

OUTPUT:

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