Loops

- The versatility of the computer lies in its ability to perform a set of instructions repeatedly.
- There are three methods by way of which we can repeat a part of a program
  - Using a **for** statement
  - Using a **while** statement
  - Using a **do-while** statement

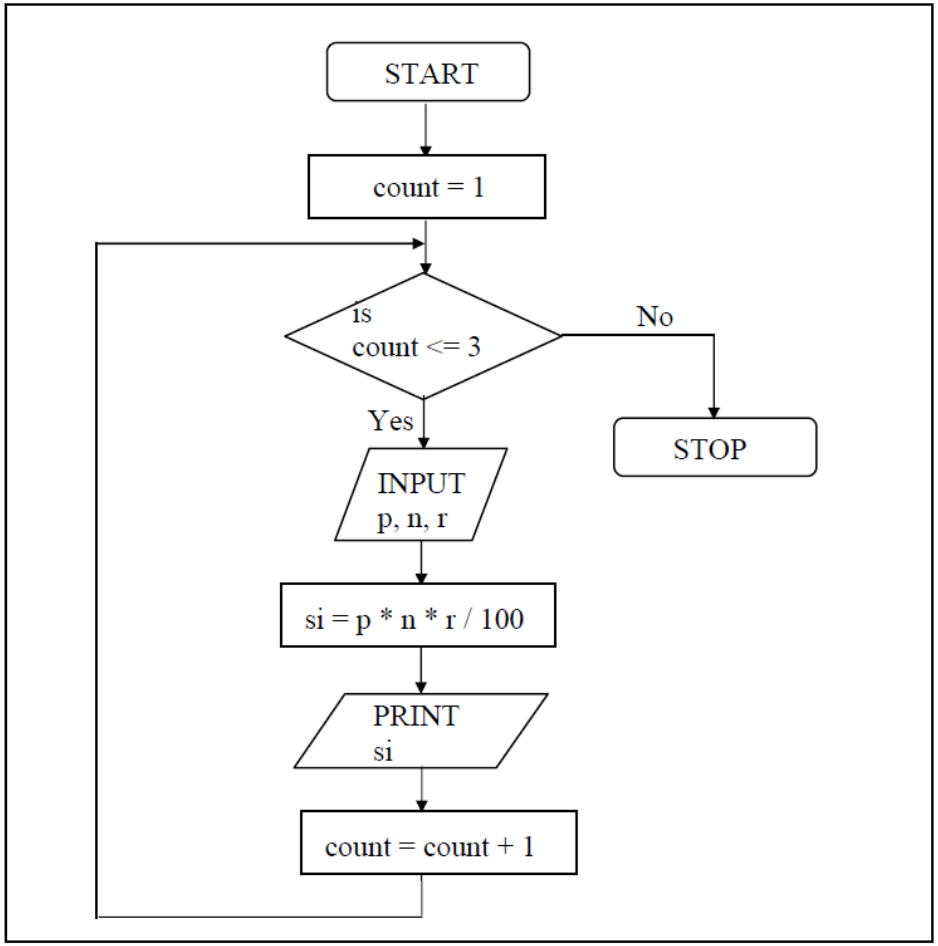
While loop

- You want to calculate gross salaries of ten different persons.

or

- You want to convert temperatures from centigrade to fahrenheit for 15 different cities

Calculation of simple interest for 3 sets of p, n and r

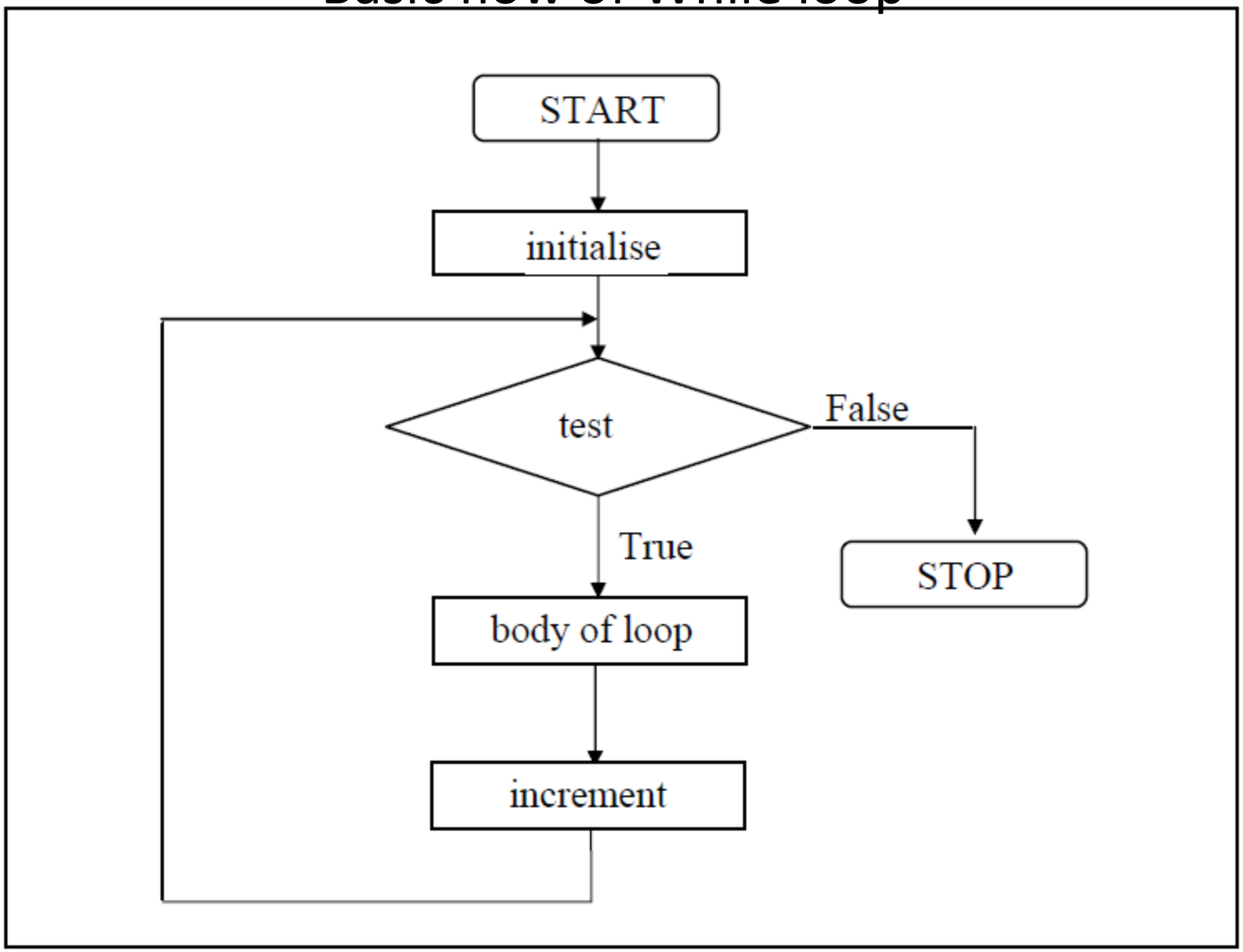


```
/* Calculation of simple interest for 3 sets of p, n and r */
main( )
{
      int   p, n, count ;
      float   r, si ;

      count = 1 ;
      while ( count <= 3 )
      {
            printf ( "\nEnter values of p, n and r " ) ;
            scanf ( "%d %d %f", &p, &n, &r ) ;
            si = p * n * r / 100 ;
            printf ( "Simple interest = Rs. %f", si ) ;

            count = count + 1 ;
      }
}
```

Output

```
Enter values of p, n and r  1000  5  13.5
Simple interest = Rs. 675.000000
Enter values of p, n and r  2000  5  13.5
Simple interest = Rs. 1350.000000
Enter values of p, n and r  3500  5  3.5
Simple interest = Rs. 612.500000
```

Basic flow of While loop



Tips and Traps

The general form of **while** is as shown below:

```
initialise loop counter ;
while ( test loop counter using a condition )
{
        do this ;
        and this ;
        increment loop counter ;
}
```

Following points about while

- The statements within the **while** loop would keep on getting executed till the condition being tested remains true.
- When the condition becomes false, the control passes to the first statement that follows the body of the **while** loop.
- In place of the condition there can be any other valid expression. So long as the expression evaluates to a non-zero value the statements within the loop would get executed.
- The condition being tested may use relational or logical operators as shown in the following examples:
  - while (i <= 10)
  - while (i >= 10 && j <= 15)
  - while (j > 10 &&(b < 15 || c < 20))

– The statements within the loop may be a single line or a block of statements. In the first case the parentheses are optional. For example,

```
while ( i <= 10 )
    i = i + 1 ;
```

is same as

```
while ( i <= 10 )
{
    i = i + 1 ;
}
```

– As a rule the while must test a condition that will eventually become false, otherwise the loop would be executed forever, indefinitely.

```
main( )
{
    int  i = 1 ;
    while ( i <= 10 )
        printf ( "%d\n", i ) ;
}
```

– Instead of incrementing a loop counter, we can even decrement it and still manage to get the body of the loop executed repeatedly. This is shown below:

```
main( )
{
    int   i = 5 ;
    while ( i >= 1 )
    {
        printf ( "\nMake the computer literate!" ) ;
        i = i - 1 ;
    }
}
```

– It is not necessary that a loop counter must only be an int. It can even be a float.

```
main( )
{
    float   a = 10.0 ;
    while ( a <= 10.5 )
    {
        printf ( "\nRaindrops on roses..." ) ;
        printf ( "...and whiskers on kittens" ) ;
        a = a + 0.1 ;
     }
 }
```

While loop

```
int main()
{
        int i=1;
        while(i<=10);
        {
                printf("%d",i);
                i=i+1;
        }
}
```

More Operators

(a)
```
main( )
{
    int   i = 1 ;
    while ( i <= 10 )
    {
        printf ( "%d\n", i ) ;
        i = i + 1 ;
    }
}
```

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

(b)
```
main( )
{
    int   i = 1 ;
    while ( i <= 10 )
    {
        printf ( "%d\n", i ) ;
        i++ ;
    }
}
```

Increment operator

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

(c)
```
main( )
{
    int   i = 1 ;
    while ( i <= 10 )
    {
        printf ( "%d\n", i ) ;
        i += 1 ;
    }
}
```

Compound Assignment operator

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

i += 1
is same as
i = i+1

- Similarly to reduce the value of a variable by 1 a decrement operator **--** is used .
- However, never use **n+++** to increment the value of **n** by 2, 
- Other compound assignment operators are **-=, *=, / =** and **%=**.
- a*=b is same as a= a*b
- a/=b is same as a= a/b
- a%=b is same as a= a%b

a= 5, b=2
a+=b will give output a=?
a-=b will give output a=?
a/=b will give output a=?
a%=b will give output a=?

- Similarly to reduce the value of a variable by 1 a decrement operator **--** is used .
- However, never use **n+++** to increment the value of **n** by 2, 
- Other compound assignment operators are **-=, *=, / =** and **%=**.
- a*=b is same as a= a*b
- a/=b is same as a= a/b
- a%=b is same as a= a%b

A= 5, b=2
a+=b will give output a=7
a-=b will give output a=3
a/=b will give output a=2
a%=b will give output a=1

Post Increment Operator

```
main( )
{
    int  i = 0 ;
    while ( i++ < 10 )
    printf ( "%d\n", i ) ;
}
```

```
0 o<10  i=1
1
1 1<10 2
2
2    2<10
3    3
```

In the statement **while (i++ < 10)**, firstly the comparison of value of **i** with 10 is performed, and then the incrementation of **i** takes place. Since the incrementation of **i** happens after its usage, here the ++ operator is called a post-incrementation operator. When the control reaches **printf( )**, **i** has already been incremented, hence **i** must be initialized to 0.

Pre Increment Operator

(e)
```
main( )
{
    int  i = 0 ;
    while ( ++i <= 10 )
        printf ( "%d\n", i ) ;
}
```

```
0
1 1<10
1
2    2<=10
2
```

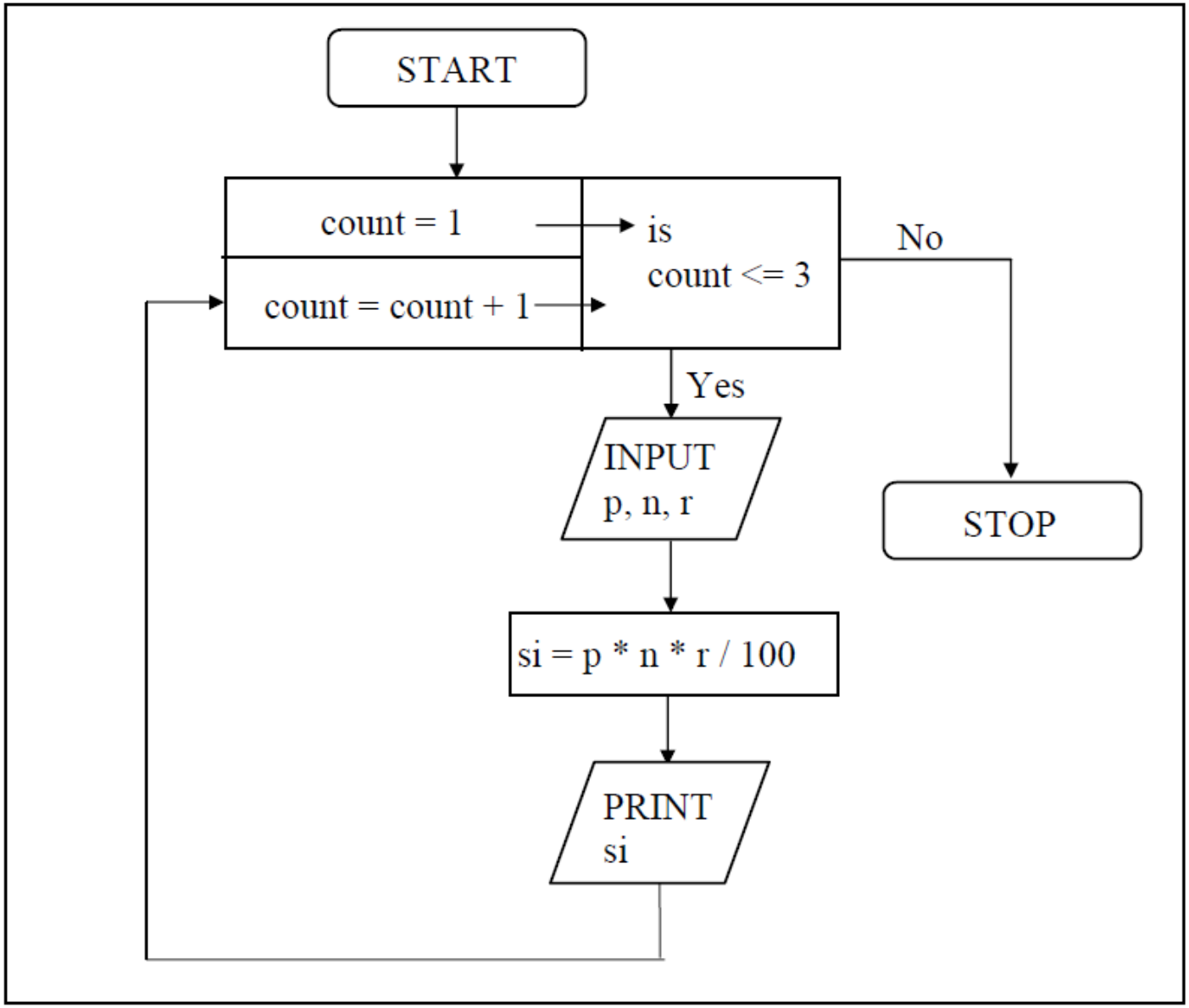
In the statement **while (++i <= 10)**, firstly incrementation of **i** takes place, then the comparison of value of **i** with 10 is performed. Since the incrementation of **i** happens before its usage, here the ++ operator is called a pre-incrementation operator.

The for loop

- **for** is the most popular looping instruction.
- The **for** allows us to specify three things about a loop in a single line

(a) Setting a loop counter to an initial value.
(b) Testing the loop counter to determine whether its value has reached the number of repetitions desired.
(c) Increasing the value of loop counter each time the program segment within the loop has been executed.

```
for ( initialise  counter ; test  counter ; increment  counter )
{
      do this ;
      and this ;
      and this ;
}
```


START
count = 1
count = count + 1
is
count <= 3
No
Yes
INPUT
p, n, r
STOP
si = p * n * r / 100
PRINT
si

```
/* Calculation of simple interest for 3 sets of p, n and r */
main ( )
{
      int   p, n, count ;
      float   r, si ;

      for ( count = 1 ; count <= 3 ; count = count + 1 )
      {
            printf ( "Enter values of p, n, and r " ) ;
            scanf ( "%d %d %f", &p, &n, &r ) ;

            si = p * n * r / 100 ;
            printf ( "Simple Interest = Rs.%f\n", si ) ;
      }
}
```

a

b

d

c

Let us now examine how the **for** statement gets executed:

- When the **for** statement is executed for the first time, the value of **count** is set to an initial value 1.

- Now the condition **count <= 3** is tested. Since **count** is 1 the condition is satisfied and the body of the loop is executed for the first time.

- Upon reaching the closing brace of **for**, control is sent back to the **for** statement, where the value of **count** gets incremented by 1.

- Again the test is performed to check whether the new value of **count** exceeds 3.

- If the value of **count** is still within the range 1 to 3, the statements within the braces of **for** are executed again.

- The body of the **for** loop continues to get executed till **count** doesn't exceed the final value 3.

- When **count** reaches the value 4 the control exits from the loop and is transferred to the statement (if any) immediately after the body of **for**.

Calculation of simple interest for 3 sets of p, n and r

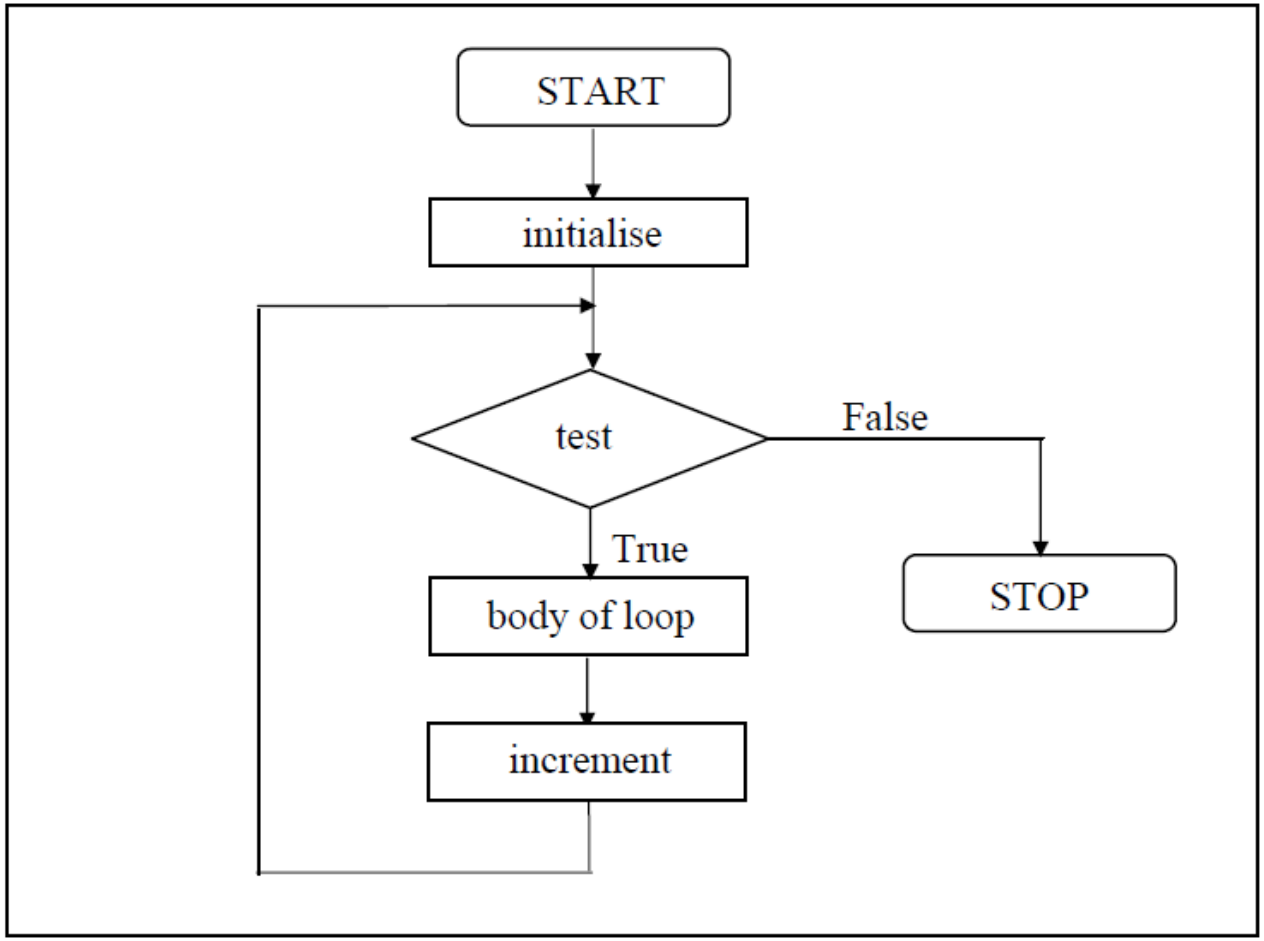
while loop

```
main( )
{
    int   p, n, count ;
    float   r, si ;

    count = 1 ;
    while ( count <= 3 )
    {
        printf ( "\nEnter values of p, n and r " ) ;
        scanf ( "%d %d %f", &p, &n, &r ) ;
        si = p * n * r / 100 ;
        printf ( "Simple interest = Rs. %f", si ) ;

        count = count + 1 ;
    }
}
```

for loop

```
main ( )
{
    int   p, n, count ;
    float   r, si ;

    for ( count = 1 ; count <= 3 ; count = count + 1 )
    {
        printf ( "Enter values of p, n, and r " ) ;
        scanf ( "%d %d %f", &p, &n, &r ) ;

        si = p * n * r / 100 ;
        printf ( "Simple Interest = Rs.%f\n", si ) ;
    }
}
```

Flowchart for the execution of the **for** loop



It is important to note that the initialization, testing and incrementation part of a **for** loop can be replaced by any valid expression. Thus the following **for** loops are perfectly ok.

```
for ( i = 10 ; i ; i -- )
      printf ( "%d", i ) ;
for ( i < 4 ; j = 5 ; j = 0 )
      printf ( "%d", i ) ;
for ( i = 1; i <=10 ; printf ( "%d",i++ )
      ;
for ( scanf ( "%d", &i ) ; i <= 10 ; i++ )
      printf ( "%d", i ) ;
```

It is important to note that the initialization, testing and incrementation part of a **for** loop can be replaced by any valid expression. Thus the following **for** loops are perfectly ok.

```
for ( i = 10 ; i ; i -- )
      printf ( "%d", i ) ;
for ( i < 4 ; j = 5 ; j = 0 )
      printf ( "%d", i ) ;
for ( i = 1; i <=10 ; printf ( "%d",i++ )
      ;
for ( scanf ( "%d", &i ) ; i <= 10 ; i++ )
      printf ( "%d", i ) ;
```

10987654321

Value of i

12345678910

print numbers from 1 to 10 in different ways using for loop

(a)
```
main( )
{
    int   i ;
    for ( i = 1 ; i <= 10 ; i = i + 1 )
        printf ( "%d\n", i ) ;
}
```

Instead of **i = i + 1**, the statements **i++** or **i += 1** can also be used.

(b)
```
main( )
{
    int   i ;
    for ( i = 1 ; i <= 10 ; )
    {
        printf ( "%d\n", i ) ;
        i = i + 1 ;
    }
}
```

incrementation is done within the body of the **for** loop and not in the **for** statement.

Note that inspite of this the semicolon after the condition is necessary.

print numbers from 1 to 10 in different ways using for loop

(c)
```
main( )
{
    int   i = 1 ;
    for ( ; i <= 10 ; i = i + 1 )
        printf ( "%d\n", i ) ;
}
```

Here the initialisation is done in the declaration statement itself,
but still the semicolon before the condition is necessary.

(d)
```
main( )
{
    int   i = 1 ;
    for ( ; i <= 10 ; )
    {
        printf ( "%d\n", i ) ;
        i = i + 1 ;
    }
}
```

Here, neither the initialisation, nor the incrementation is done in the **for** statement,

but still the two semicolons are necessary.

print numbers from 1 to 10 in different ways using for loop

(e)
```
main( )
{
    int  i ;
    for ( i = 0 ; i++ < 10 ; )
        printf ( "%d\n", i ) ;
}
```

Output: 12345678910

the comparison as well as the incrementation is done through the same statement, **i++ < 10**.
Since the ++ operator comes after i firstly comparison is done, followed by incrementation.
Note that it is necessary to initialize **i** to 0.

(f)
```
main( )
{
    int  i ;
    for ( i = 0 ; ++i <= 10 ; )
        printf ( "%d\n", i ) ;
}
```

Output: 12345678910

Here, both, the comparison and the incrementation is done through the same statement, **++i <= 10**.
Since ++ precedes i firstly incrementation is done, followed by comparison.
Note that it is necessary to initialize **i** to 0.