# Loop Control Structure

## Introduction

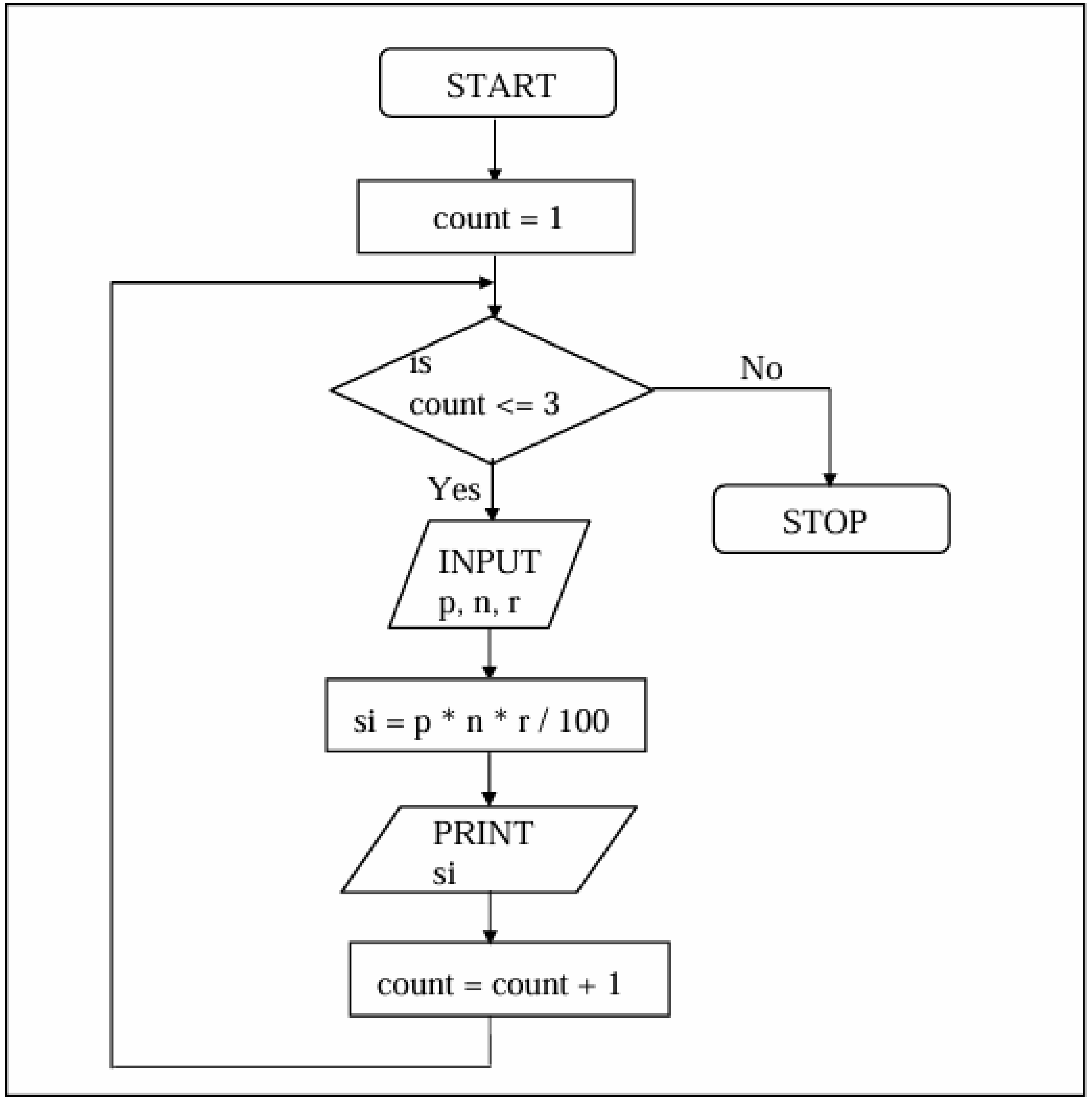
- So far used either a sequential or a decision control instruction.
- In the first one, the calculations were carried out in a fixed order,
- while in the second, an appropriate set of instructions were executed depending upon the outcome of the condition being tested
- we frequently need to perform an action over and over, often with variations in the details each time.
- The mechanism, which meets this need, is the ‘loop’

## Loops

- The versatility of the computer lies in its ability to perform a set of instructions repeatedly.
- This involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied.
- This repetitive operation is done through a loop control instruction
- 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

## The while Loop

- It is often the case in programming that you want to do something a fixed number of times.
- Perhaps you want to calculate gross salaries of ten different persons, or you want to convert temperatures from centigrade to fahrenheit for 15 different cities.
- The while loop is ideally suited for such cases


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

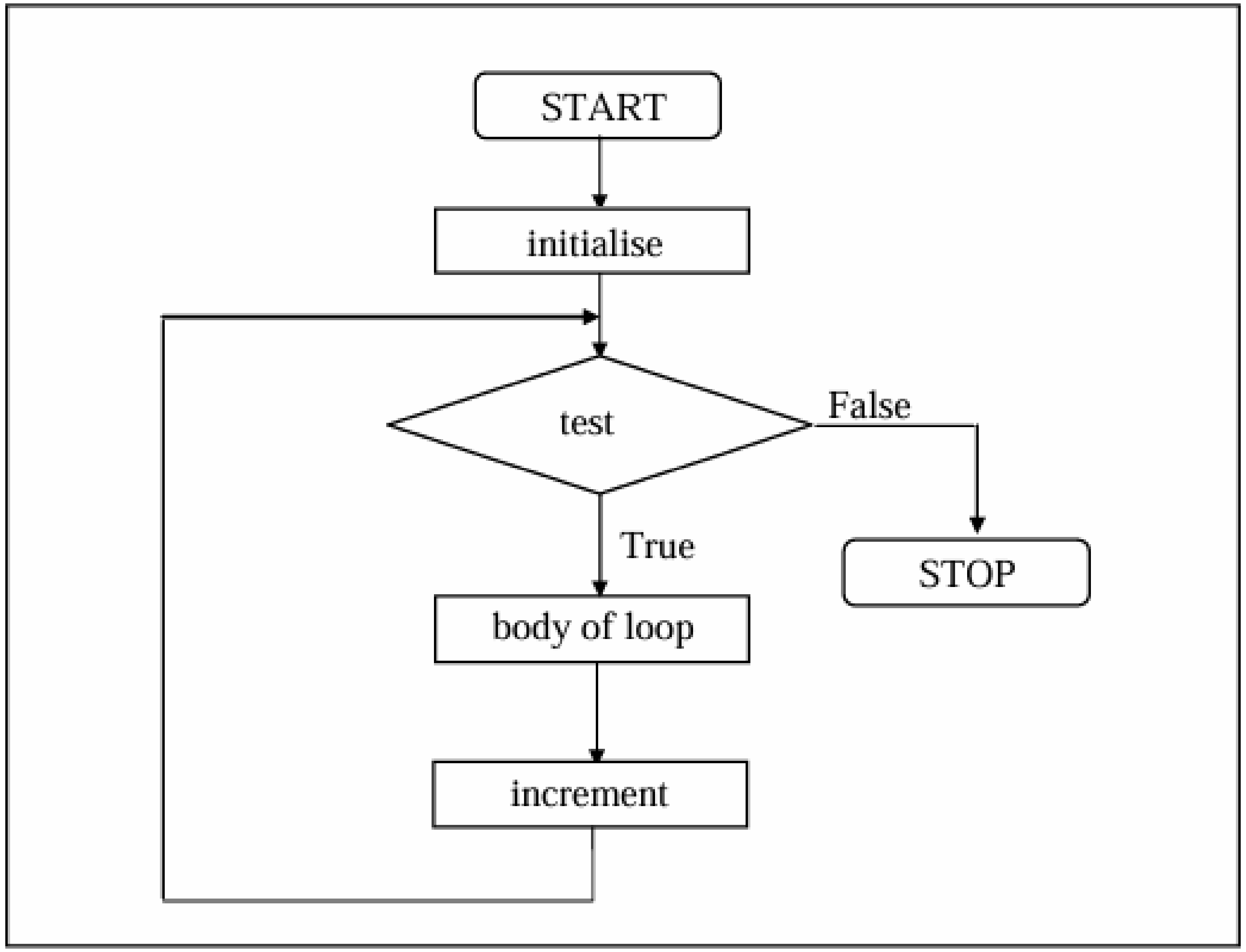
```
/* Calculation of simple interest for 3 sets of p, n and r */
main( )
{
    int   p, n, count ;
    float   r, si ;

    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 ;
    }
}
```

```
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
```


START
initialise
test
False
True
STOP
body of loop
increment

## Points to note

- 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 loop may be a single line or block of statements, in the first case the parentheses are optional.
- while ( i <= 10 )

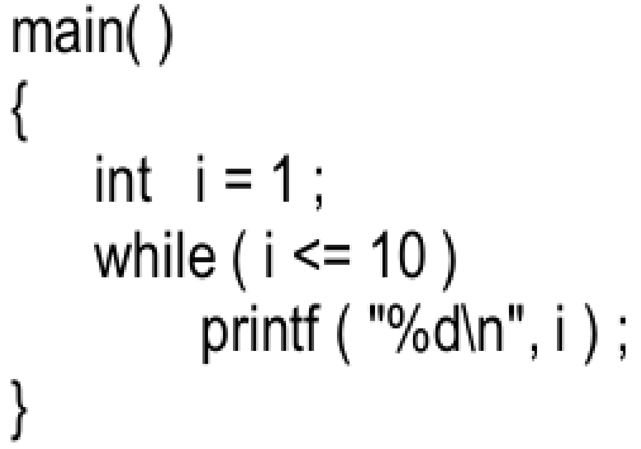
```
        i = i + 1 ;
```

is same as:

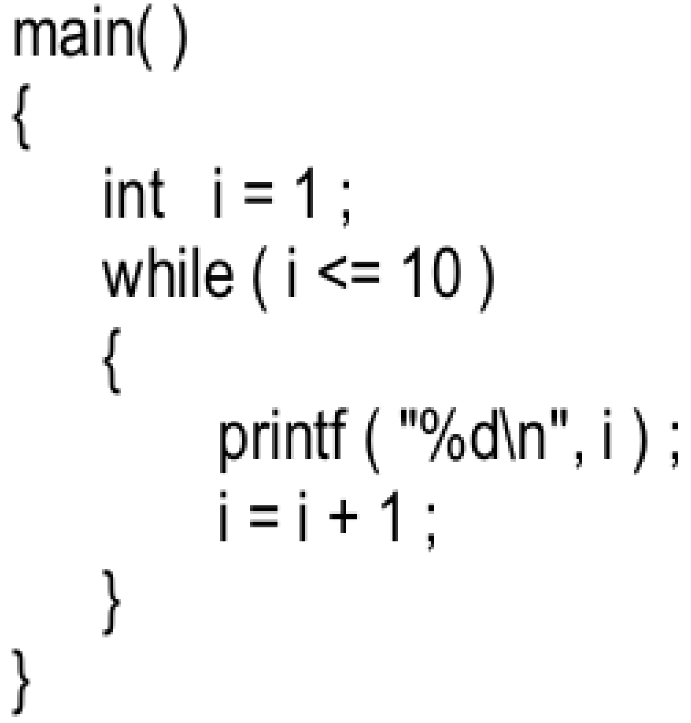
- while ( i <= 10)

```
         { 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
- For example, below is an indefinite loop



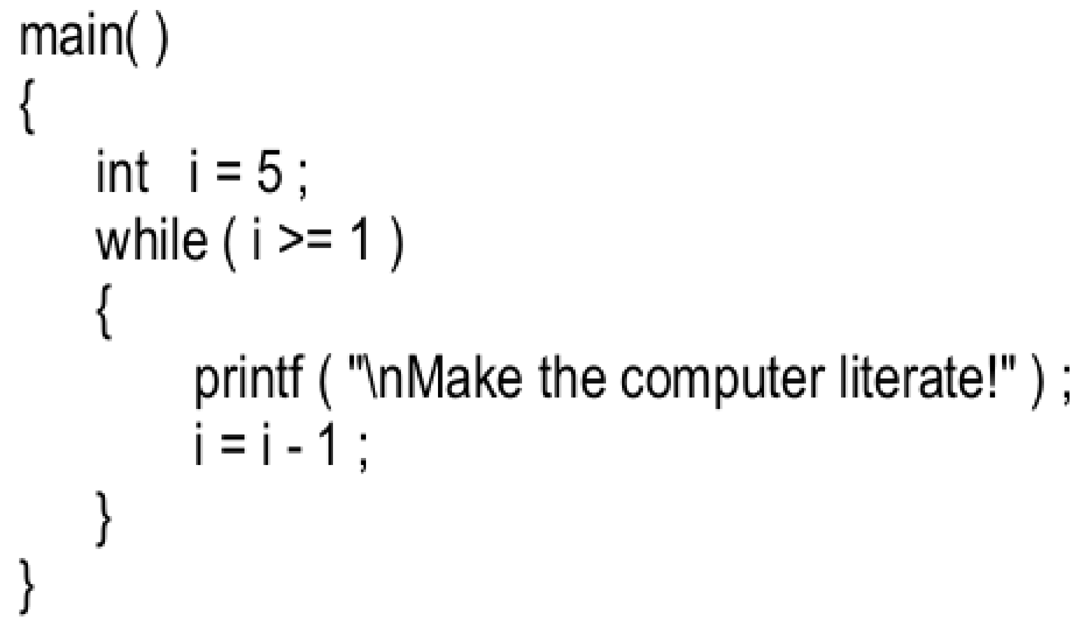
```
main( )
{
    int  i = 1 ;
    while ( i <= 10 )
        printf ( "%d\n", i ) ;
}
```



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

# Decrementing the loop counter

- Instead of incrementing a loop counter, we can even decrement



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

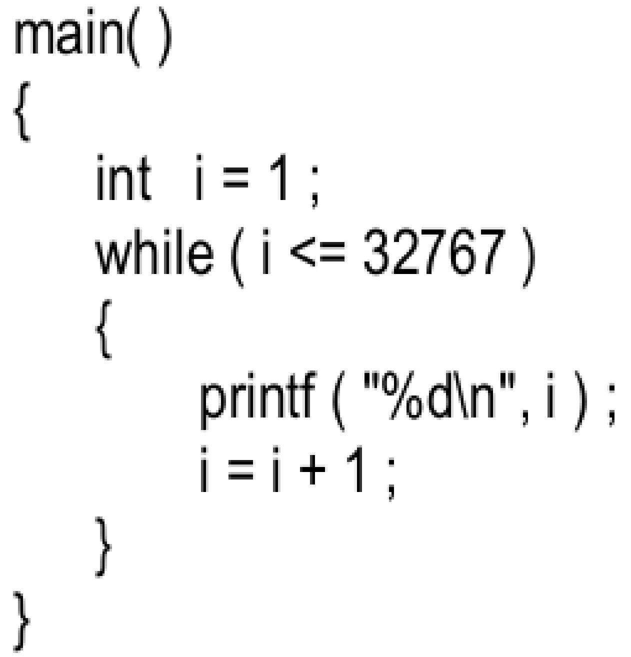
## Float as loop counter

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


       }
  }
```

## Indefinite loop

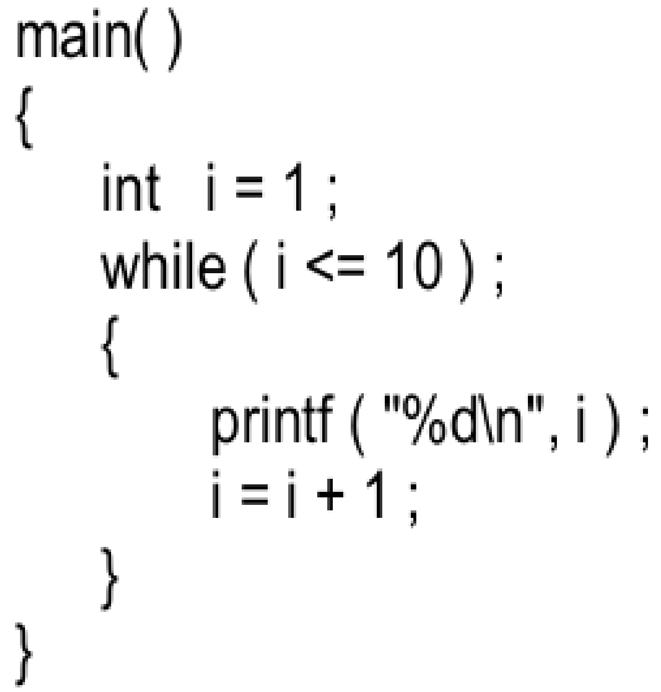
- After 32767, value of becomes -32768 thus getting into an indefinite loop



```
main( )
{
    int   i = 1 ;
    while ( i <= 32767 )
    {
        printf ( "%d\n", i ) ;
        i = i + 1 ;
    }
}
```

# Indefinite loop

- ; after while makes it an infinite loop
- enclosing printf( ) and i = i +1 within a pair of braces is not an error. In fact we can put a pair of braces around any individual statement or set of statements without affecting the execution of the program



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

# More Operators

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

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

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

(d)
```
main( )
{
    int  i = 0 ;
    while ( i++ < 10 )

            printf ( "%d\n", i ) ;
 }
```

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

## The for Loop

- the most popular looping instruction
- The for allows us to specify three things about a loop in a single line
- Setting a loop counter to an initial value.
- Testing the loop counter to determine whether its value has reached the number of repetitions desired.
- Increasing the value of loop counter each time the program segment within the loop has been executed.

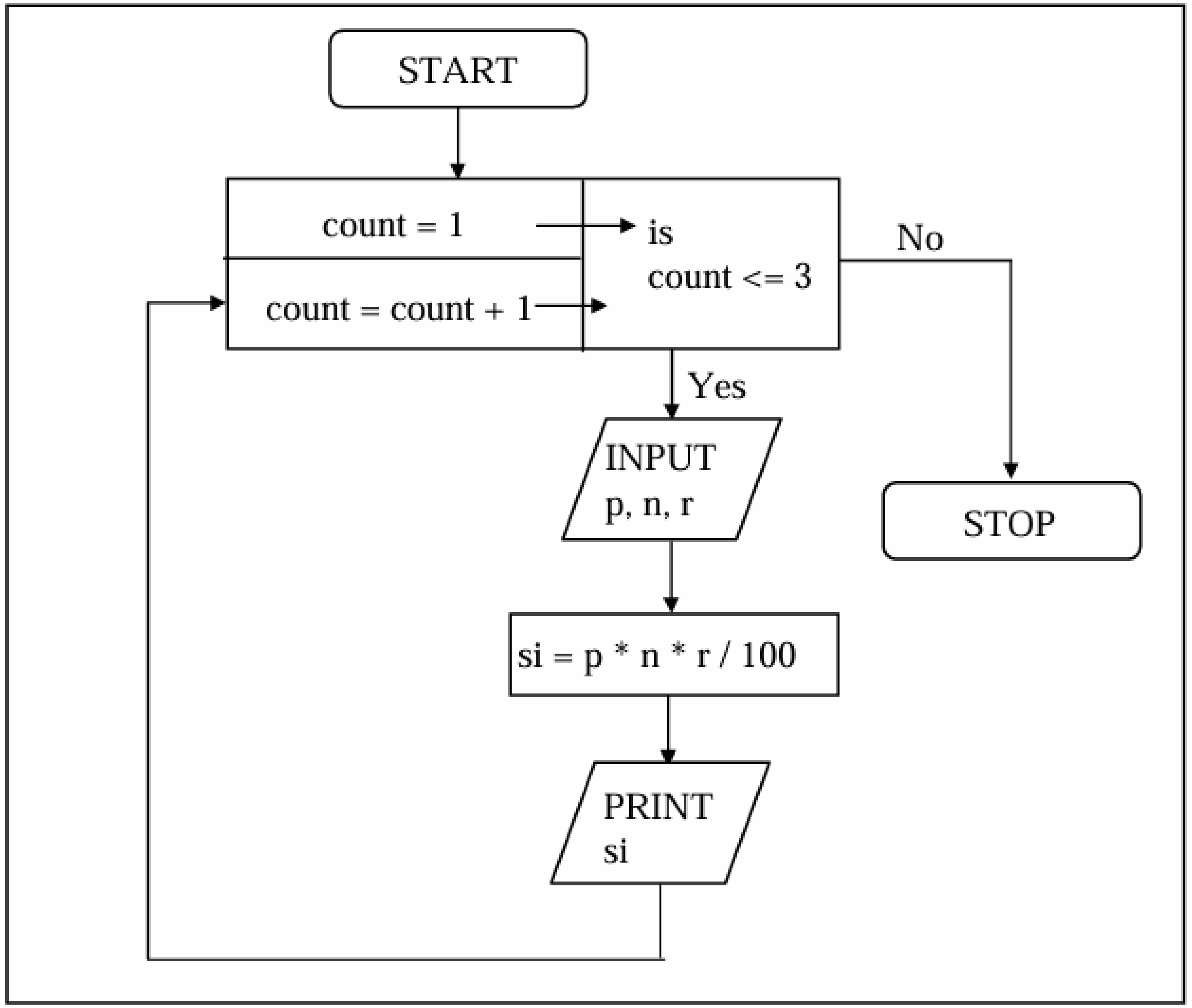
## The general form of for

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

```
/* 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 ) ;
    }
}
```


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

## Points to note

- Note that the initialization, testing and incrementation part of a for loop can be replaced by any valid expression
- Thus all of the following are 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 ) ;

## Different ways of writing the same code

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

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

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

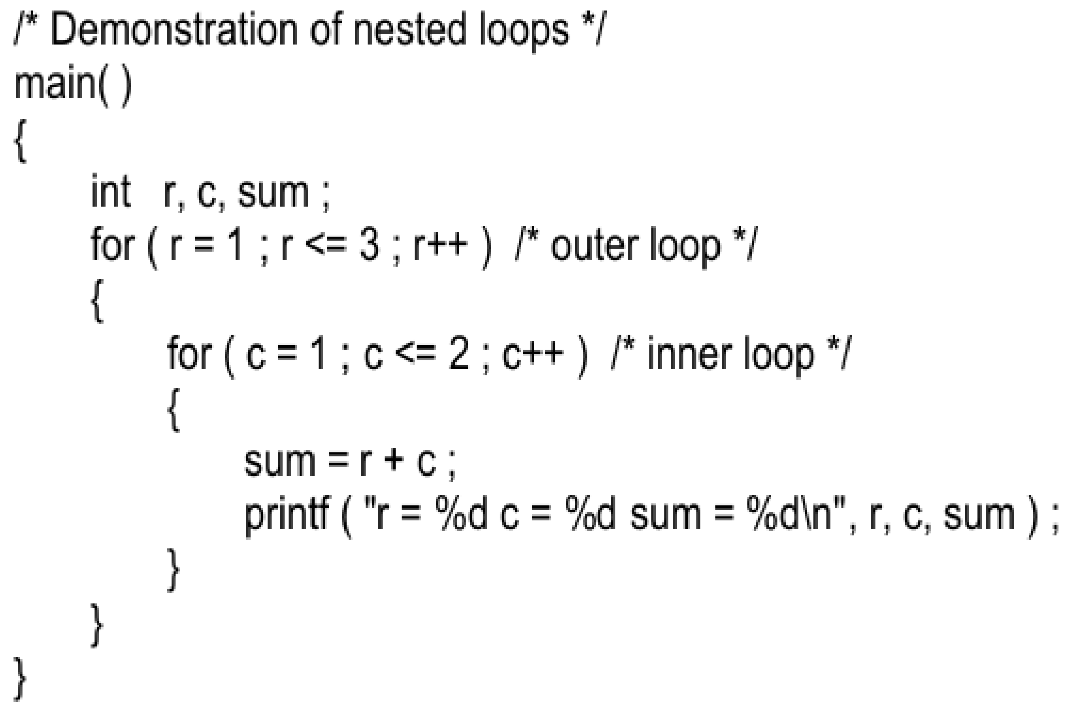
(d)
```
main( )
{
    int  i = 1 ;
    for ( ; i <= 10 ; )
    {
        printf ( "%d\n", i ) ;
        i = i + 1 ;
    }
}
```

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

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

## Nesting of Loops

- The way if statements can be nested, similarly whiles and fors can also be nested.

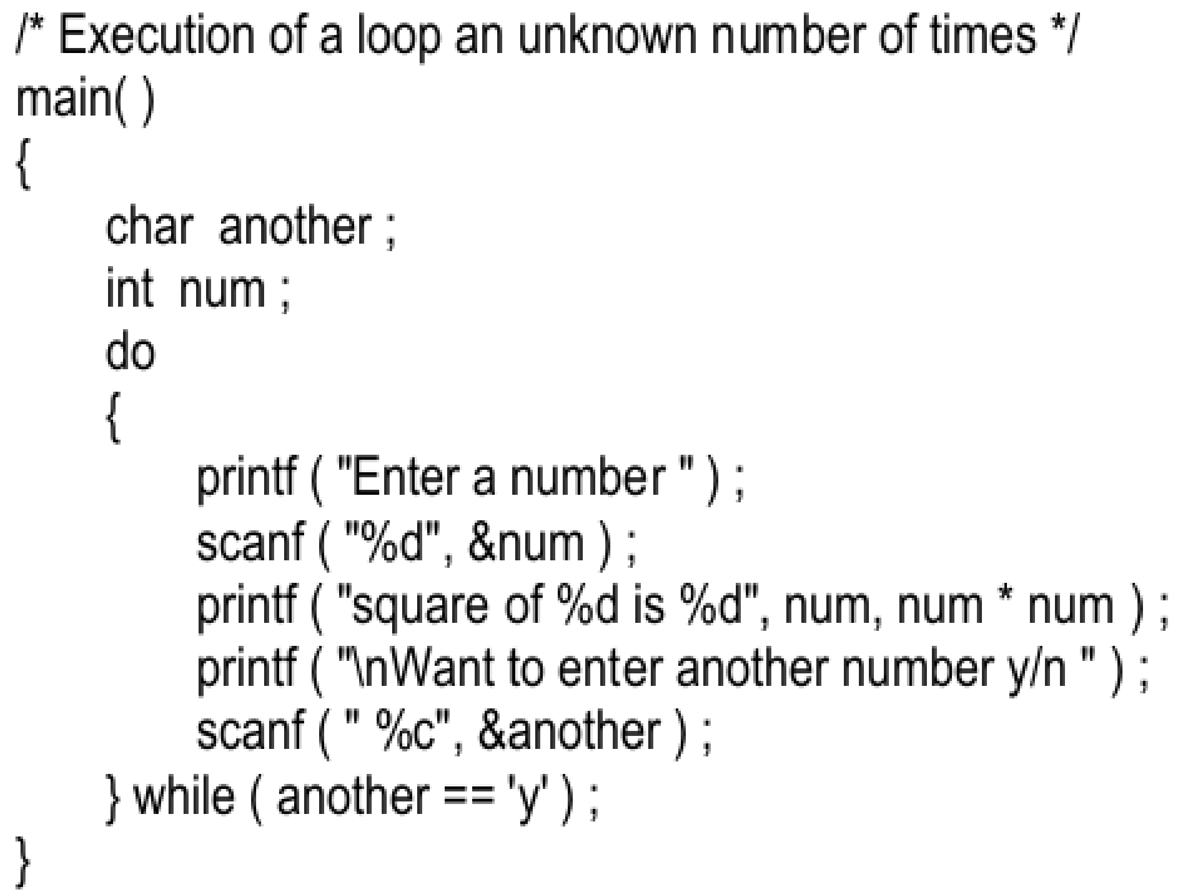


```
/* Demonstration of nested loops */
main( )
{
	int   r, c, sum ;
	for ( r = 1 ; r <= 3 ; r++ )  /* outer loop */
	{
		for ( c = 1 ; c <= 2 ; c++ )  /* inner loop */
		{
			sum = r + c ;
			printf ( "r = %d c = %d sum = %d\n", r, c, sum ) ;
		}
	}
}
```

## Multiple Initialisations in the for Loop

- The initialisation expression of the for loop can contain more than one statement separated by a comma. For example,
- for ( i = 1, j = 2 ; j <= 10 ; j++ )
- Multiple statements can also be used in the incrementation expression of for loop; i.e., you can increment (or decrement) two or more variables at the same time
- However, only one expression is allowed in the test expression. This expression may contain several conditions linked together using logical operators.

## When number of times the loop is to be executed is not known



```
/* Execution of a loop an unknown number of times */
main( )
{
	char  another ;
	int  num ;
	do
	{
		printf ( "Enter a number " ) ;
		scanf ( "%d", &num ) ;
		printf ( "square of %d is %d", num, num * num ) ;
		printf ( "\nWant to enter another number y/n " ) ;
		scanf ( " %c", &another ) ;
	} while ( another == 'y' ) ;
}
```

```c
/* odd loop using a for loop */
main( )
{
	char  another = 'y' ;
	int  num ;
	for ( ; another == 'y' ; )
	{
		printf ( "Enter a number " ) ;
		scanf ( "%d", &num ) ;
		printf ( "square of %d is %d", num, num * num ) ;
		printf ( "\nWant to enter another number y/n " ) ;
		scanf ( " %c", &another ) ;
	}
}
```

```c
/* odd loop using a while loop */
main( )
{
	char  another = 'y' ;
	int  num ;

	while ( another == 'y' )
	{
		printf ( "Enter a number " ) ;
		scanf ( "%d", &num ) ;
		printf ( "square of %d is %d", num, num * num ) ;
		printf ( "\nWant to enter another number y/n " ) ;
		scanf ( " %c", &another ) ;
	}
}
```

## The break Statement

- We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test.
- The keyword break allows us to do this.
- When break is encountered inside any loop, control automatically passes to the first statement after the loop.
- A break is usually associated with an if.
- The keyword break, breaks the control only from the loop in which it is placed

## Example

- Write a program to determine whether a number is prime or not. A prime number is one, which is divisible only by 1 or itself.

## Solution

```
main( )
{
     int   num, i ;

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

     i = 2 ;
     while ( i <= num - 1 )
     {
          if ( num % i == 0 )
          {
               printf ( "Not a prime number" ) ;
               break ;
          }
          i++ ;
     }


       if ( i == num )
            printf ( "Prime number" ) ;
 }
```

## Control breaks only from the current block

```
main( )
{
     int  i = 1 , j = 1 ;

     while ( i++ <= 100 )
     {
          while ( j++ <= 200 )
          {
               if ( j == 150 )
                    break ;
               else
                    printf ( "%d %d\n", i, j ) ;
          }


        }
  }
```

## The continue Statement

- In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed.
- The keyword continue allows us to do this.
- When continue is encountered inside any loop, control automatically passes to the beginning of the loop
- A continue is usually associated with an if

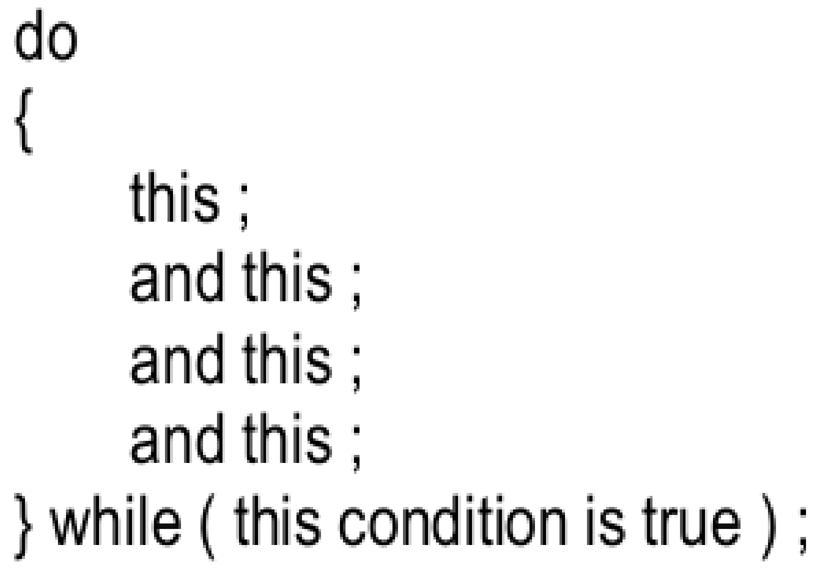
```
main( )
{
	int   i, j ;

	for ( i = 1 ; i <= 2 ; i++ )
	{
		for ( j = 1 ; j <= 2 ; j++ )
		{
			if ( i == j )
				continue ;

			printf ( "\n%d %d\n", i, j ) ;
		}
	}
}
```

## The do-while Loop

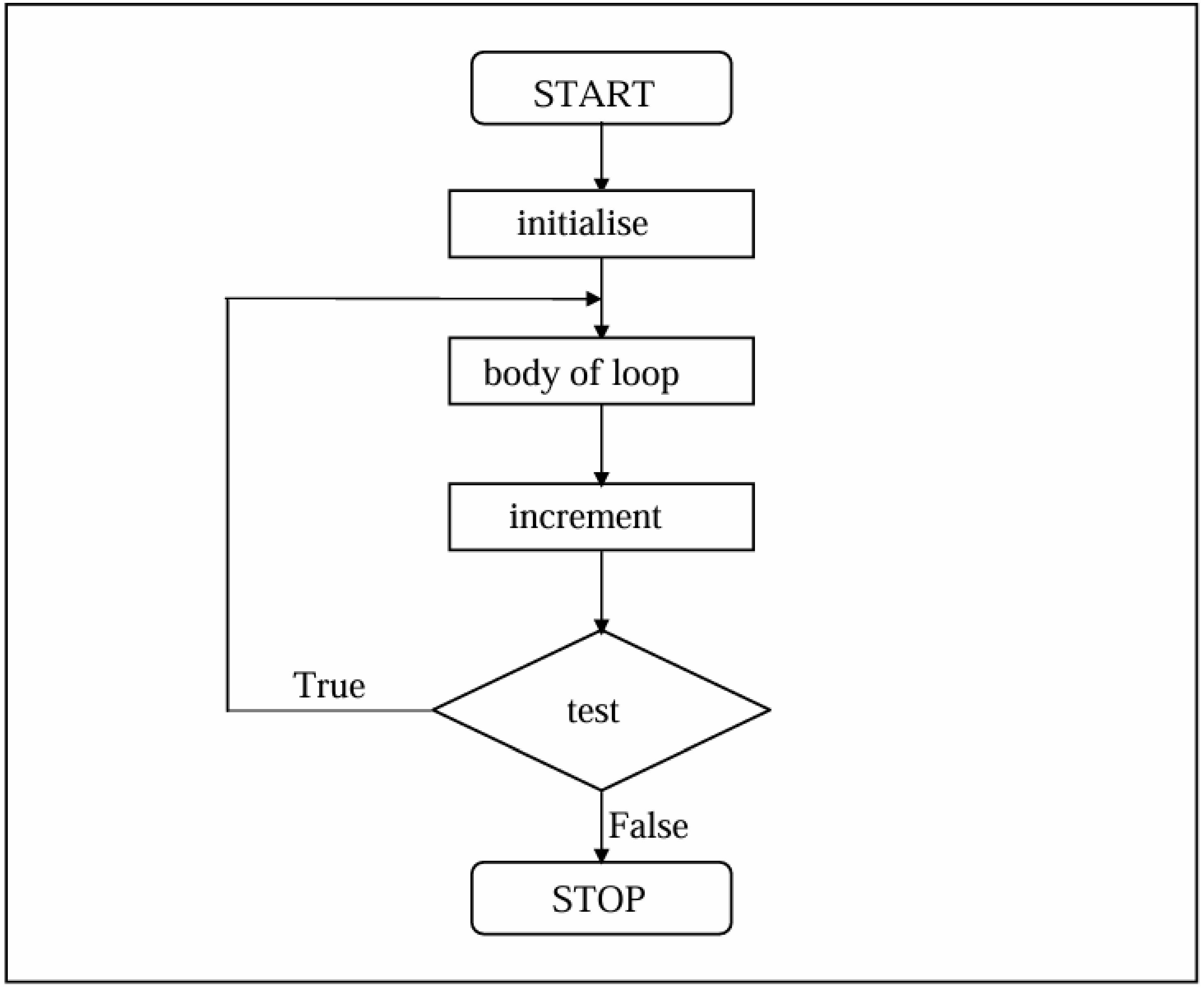
- The do-while loop looks like this:



```
do
{
        this ;
        and this ;
        and this ;
        and this ;
} while ( this condition is true ) ;
```

## Difference between while and do-while

- There is a minor difference between the working of while and do while loops.
- This difference is the place where the condition is tested.
- The while tests the condition before executing any of the statements within the while loop.
- As against this, the do-while tests the condition after having executed the statements within the loop
- This means that do-while would execute its statements at least once, even if the condition fails for the first time


START
initialise
body of loop
increment
test
True
False
STOP

```
main( )
{
        while ( 4 < 1 )
                printf ( "Hello there \n") ;
}
```

```
main( )
{
        do
        {
                printf ( "Hello there \n") ;
        } while ( 4 < 1 ) ;
}
```

## Important Points

- break and continue are used with do-while just as they would be in a while or a for loop.
- A break takes you out of the do-while bypassing the conditional test.
- A continue sends you straight to the test at the end of the loop.