



**Syrian Private University**

# **Introduction to Algorithms and Programming**

**Instructor: Dr. Mouhib Alnoukari**

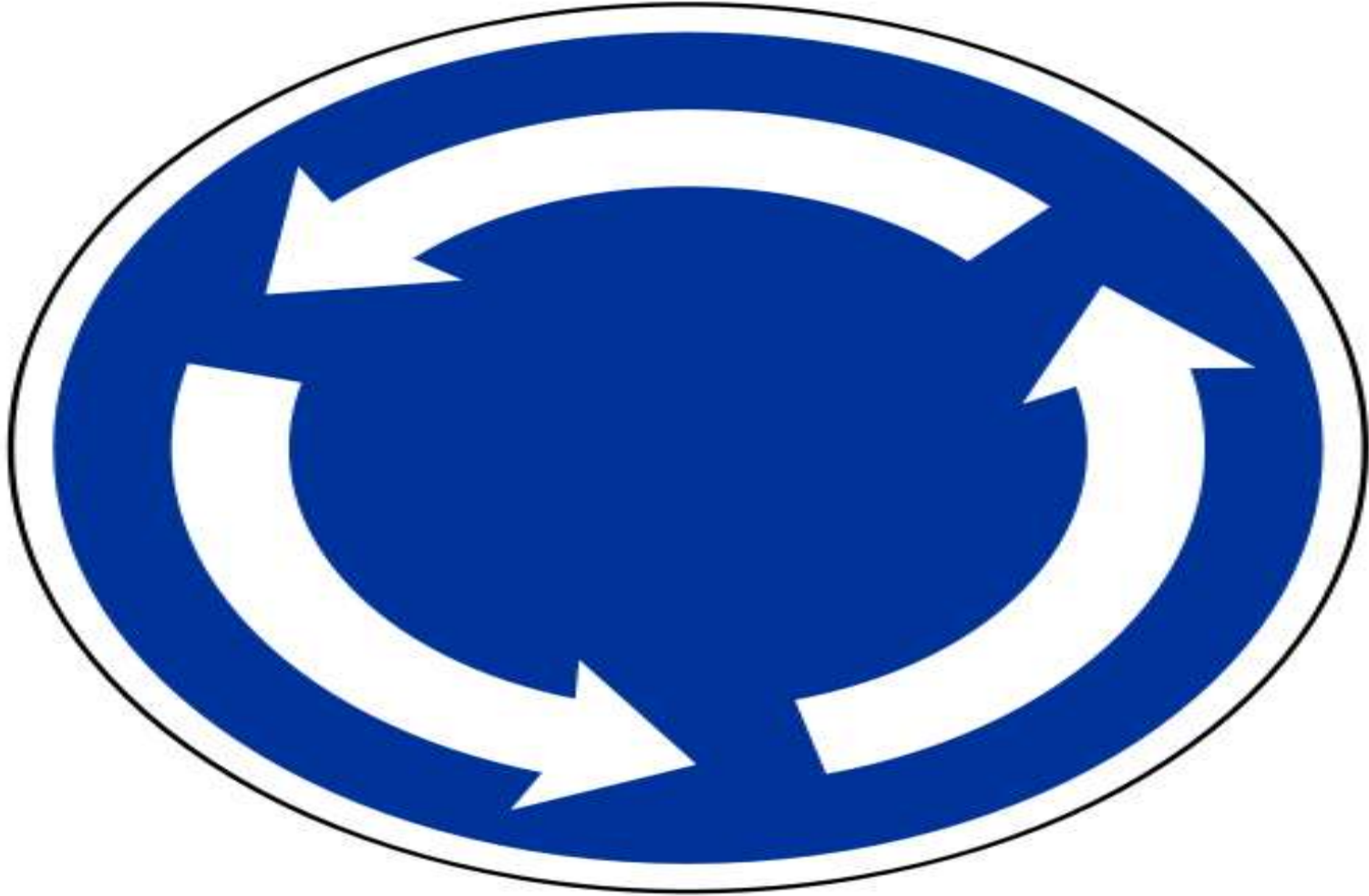


# Loops



# Loop analogy (roundabout)

---





# Loop



# Exiting a Loop



# Repetition Statements

---

- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- C has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*
- The programmer should choose the right kind of loop for the situation



# Loop constructs in C

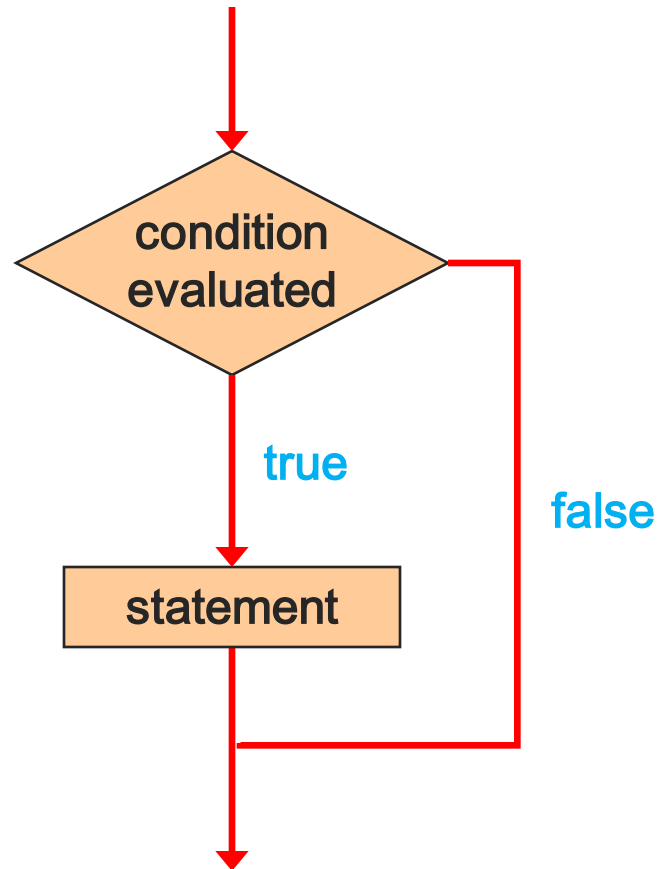
---

Loops = repetition statements

1. do-while loop (or do loop for short)
2. while loop
3. for loop

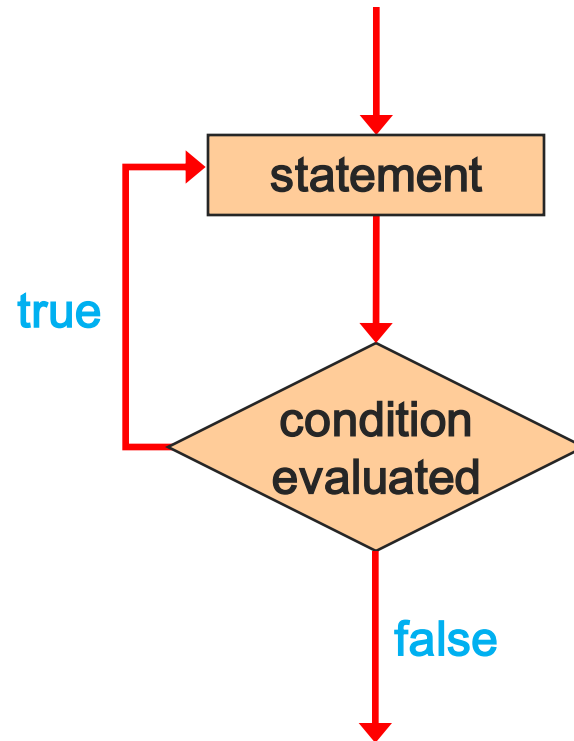


# Logic of an if statement





# Logic of a do Loop



# The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement;
}
while ( condition );
```

- The *statement* is executed once initially, and then the *condition* is evaluated
- The statement is executed repeatedly until the condition becomes false

# The do Statement

---

- An example of a `do` loop:

```
int count = 0;
do
{
    count++;
    printf("%d\n", count);
} while (count < 5);
```

- The body of a `do` loop is executed at least once



# Example: Fixing Bad Keyboard Input

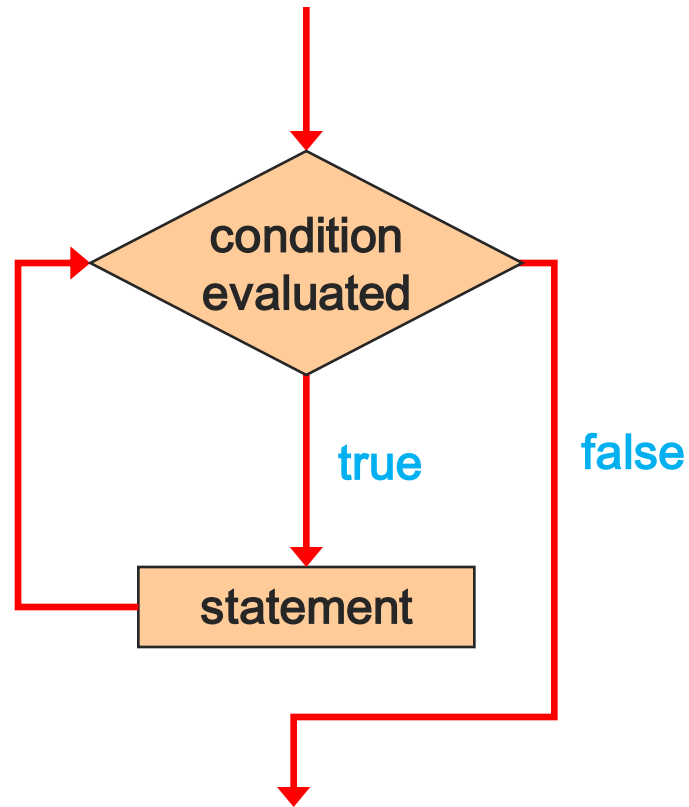
---

- Write a program that refuses to accept a negative number as an input.
- The program must keep asking the user to enter a value until he/she enters a positive number.
- How can we do this?





# Logic of a while Loop



# The while Statement

---

- A *while statement* has the following syntax:

```
while ( condition )  
    statement;
```

- If the *condition* is true, the *statement* is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false



# The while Statement

---

- An example of a while statement:

```
int count = 1;
while (count <= 5)
{
    printf ("%d\n", count);
    count++;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Therefore, the body of a `while` loop will execute zero or more times



# The while Statement

---

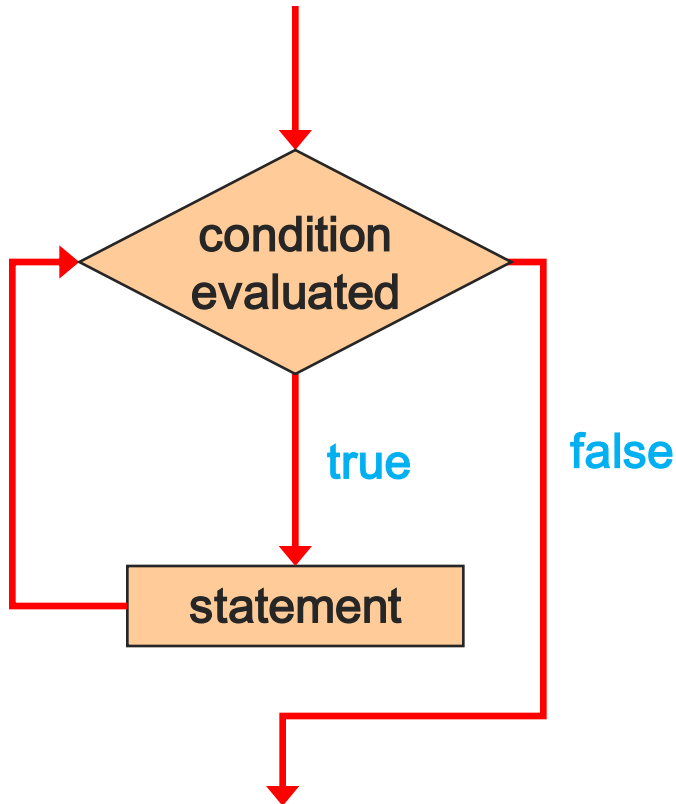
- Let's look at some examples of loop processing
- A loop can be used to maintain a *running sum*
- A *sentinel value* is a special input value that represents the end of input
- A loop can also be used for *input validation*, making a program more *robust*



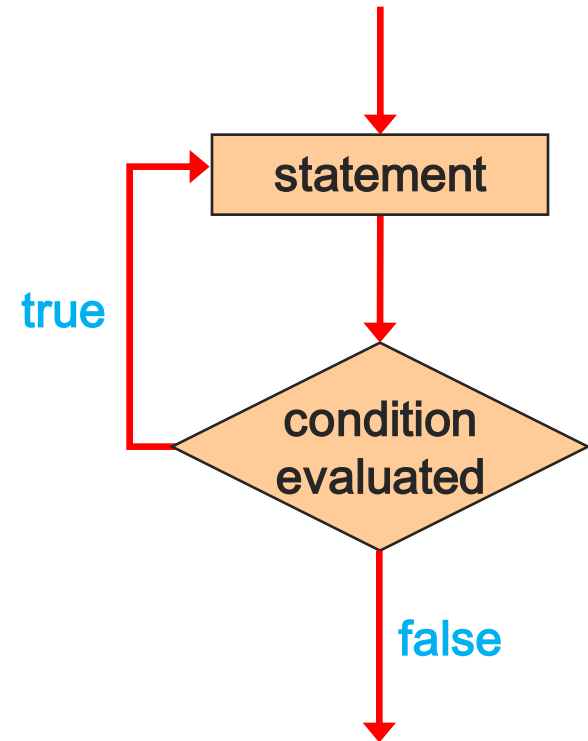


# Comparing while and do

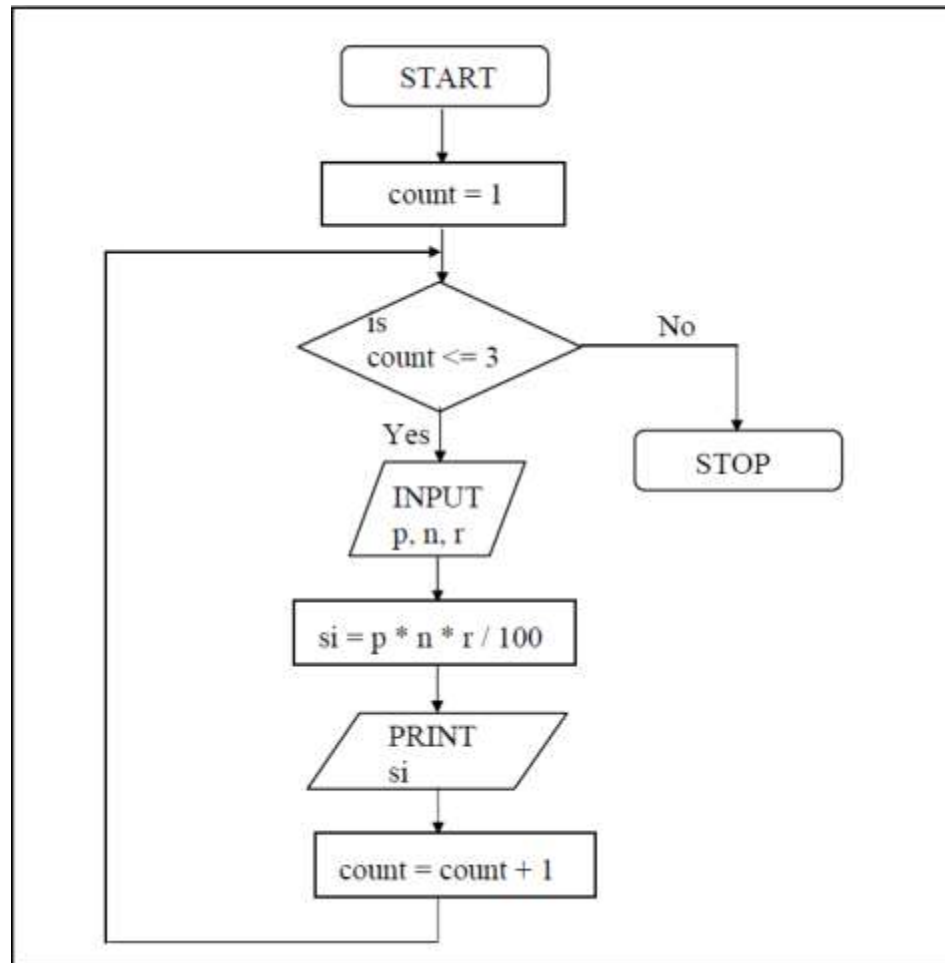
## The while Loop



## The do Loop



# Example: Calculation of simple interest for 3 sets



# Example: Calculation of simple interest for 3 sets

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

    count = 1 ;

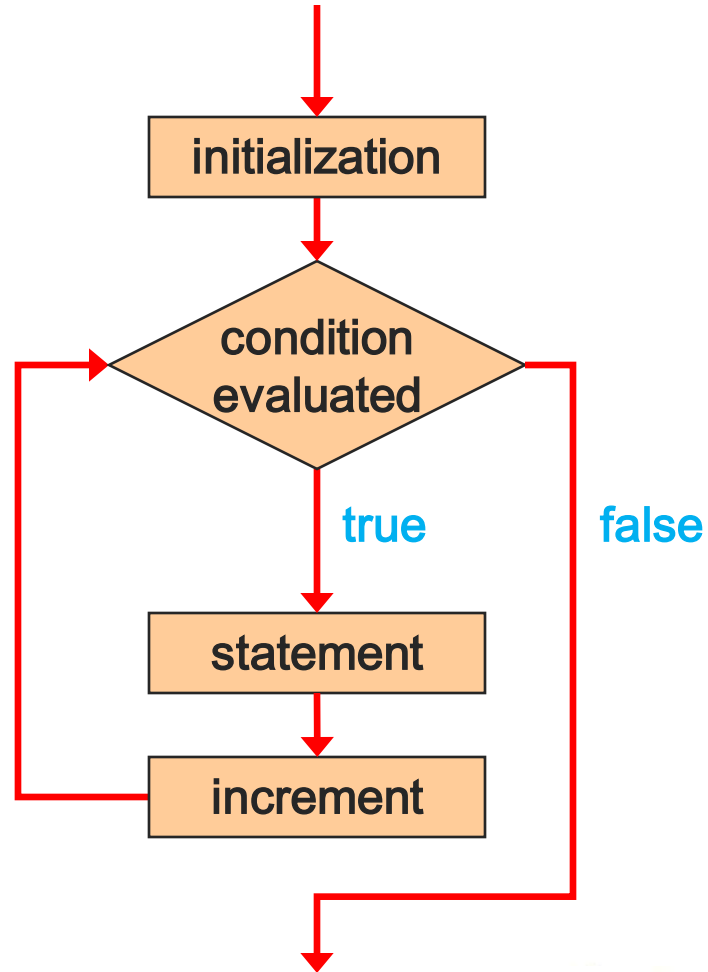
    while ( count <= 3 )
    {
        printf ( "\nEnter values of p, n and r " ) ;
        scanf ( "%d %d %f", &p, &n, &r ) ;
        si = p * n * r / 100 ;
        printf ( "Simple interest = Rs. %f", si ) ;

        count = count + 1 ;
    }
}
```

And here are a few sample runs...

```
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
```

# Logic of a for loop





# The for Statement

- A *for statement* has the following syntax:

The *initialization*  
is executed once  
before the loop begins

The *statement* is  
executed until the  
*condition* becomes false

for ( *initialization* ; *condition* ; *increment* )  
    *statement*;

The *increment* portion is executed at  
the end of each iteration

# The for Statement

---

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;  
while ( condition )  
{  
    statement;  
    increment;  
}
```

# The for Statement

---

- An example of a `for` loop:

```
for (int count=1; count <= 5; count++)  
    printf ("%d\n", count);
```

- The initialization section can be used to declare a variable
- Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body
- Therefore, the body of a `for` loop will execute zero or more times



# The for Statement

---

- The increment section can perform any calculation

```
int num;  
for (num=100; num > 0; num -= 5)  
    printf ("%d\n", num);
```

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance





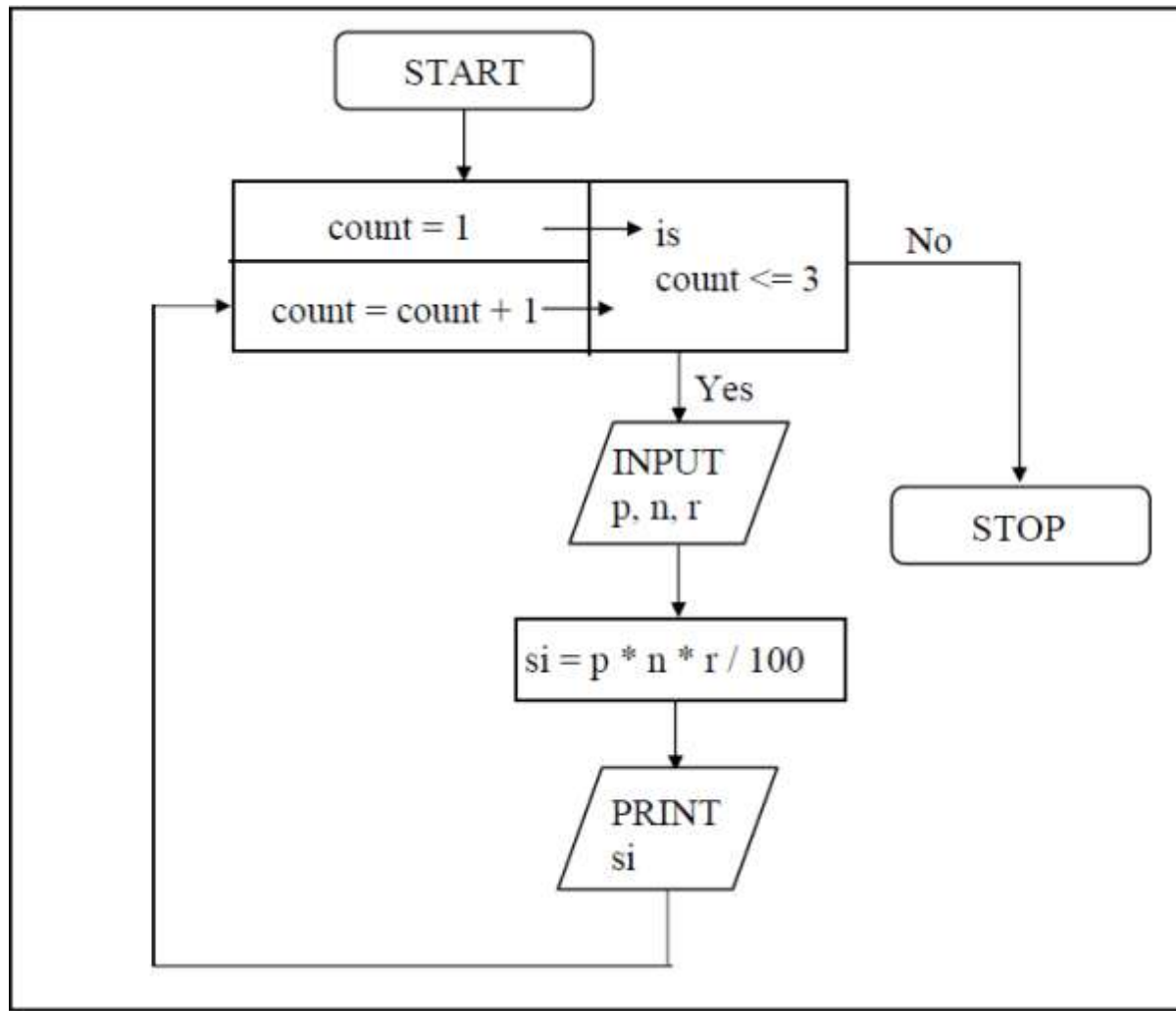
# The for Statement

---

- Each expression in the header of a `for` loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed



# Example: Calculation of simple interest for 3 sets



# Example: Calculation of simple interest for 3 sets

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

# Infinite Loops

---

- The body of a `while` loop eventually must make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally



# Infinite Loops

---

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
    printf ("%d\n", count);
    count = count - 1;
}
```

- This loop will continue executing until interrupted (Control-C) or until an underflow error occurs



# Nested Loops

---

- Similar to nested `if` statements, loops can be nested as well.
- That is, the body of a loop can contain another loop.
- For each iteration of the outer loop, the inner loop iterates completely.





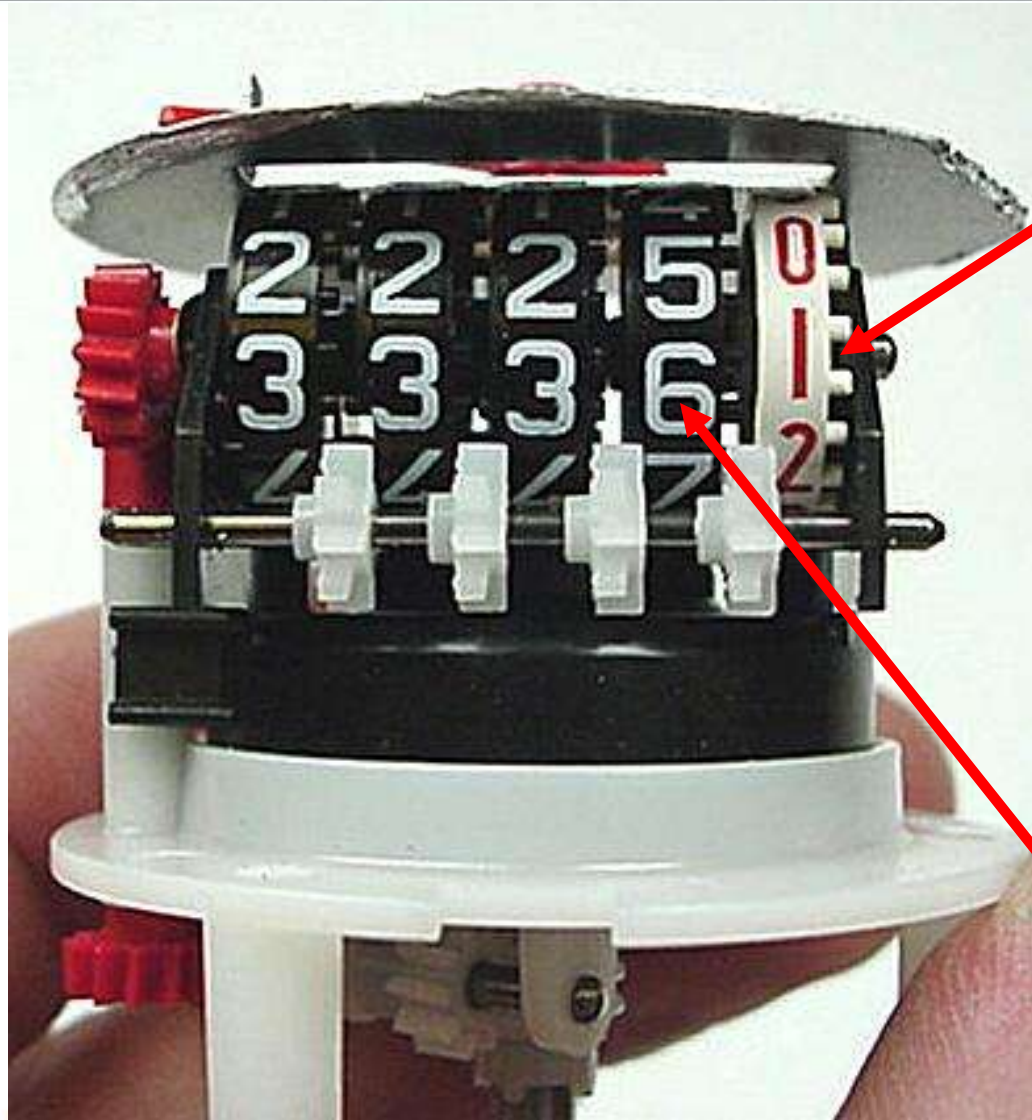
# Nested Loops

- How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 <= 20)
    {
        printf ("Here");
        count2++;
    }
    count1++;
}
```

**10 \* 20 = 200**

# Analogy for Nested Loops



**Inner Loop**

**Outer Loop**

# Example: Stars

- Write a program that prints the following

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



# Example

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

```
r = 1 c = 1 sum = 2
r = 1 c = 2 sum = 3
r = 2 c = 1 sum = 3
r = 2 c = 2 sum = 4
r = 3 c = 1 sum = 4
r = 3 c = 2 sum = 5
```

# Exercises

- (a) Write a program to calculate overtime pay of 10 employees. Overtime is paid at the rate of Rs. 12.00 per hour for every hour worked above 40 hours. Assume that employees do not work for fractional part of an hour.
- (b) Write a program to find the factorial value of any number entered through the keyboard.
- (c) Two numbers are entered through the keyboard. Write a program to find the value of one number raised to the power of another.