For Loop

- Let's discuss repeating code.
    - ➔ The C programming language has a few constructs specifically designed to handle these situations when you need to use the same code repeatedly.
    - ➔ You can repeat a block of statements until some condition is met or a specific number of times repeating code without a condition is a forever/infinite loop.
- The number of times that a loop is repeated can be controlled simply by a count.
    - ➔ Repeating the statement block a given number of time(counter controlled loop).
- The number of times that a loop is repeated can depend on when a condition is met.
    - ➔ The user entering “quit”.
- You typically use the for loop to execute a block of statements a given number of times.
- If you want to display the numbers from 1 to 10
    - ➔ Instead of writing ten statements that call printf(), you would use a for loop.

```
for(int count = 1 ; count >= 10; ++count)
{
printf)”%d”, count);
}
```

- The for loop operation is controlled by what appears between the parentheses that follow the keyword for.
    - ➔ The three control expressions that are separated by semicolons control the operation of the loop
- The action that you want to repeat each time loop repeats is the block containing the statement that calls printf() (body of the loop).
    - ➔ For single statements, you can omit the braces

For Syntax

- The general pattern for the for loop is :

```
for (starting_condition ; continuation_condition ;
action_per_iteration)
loop_statement;
```

- The statement to be repeated is represented by loop_statement
    - ➔ Could equally well be a block of several statements enclosed between braces.

starting_condition

- The **`starting_condition`** usually (but not always) sets an initial value to a loop control variable.
    - ➔ The loop control variable is typically a counter of some kind that tracks how often the loop has been repeated.
    - ➔ Can also declared and initialized several variables of the same type here with the declarations separated by commas.
    - ➔ Variables will be local to the loop and will not exist once the loop ends.

continuation_conditon

- The **`continuation_condition`** is a logical expression evaluating to true or false.
  - ➔ Determines whether the loop should continue to be executed .
  - ➔ As long as this condition has the value true, the loop continues,
  - ➔ Typically checks the values of the loop control variable.
  - ➔ You can put any logical or arithmetic expression here as long as you know what you are doing.
  - ➔ Is tested at the beginning of the loop rather than at the end. Means that that the loop_statement will not be executed at all if the continuation_condition starts out as false.

action_per_iteration

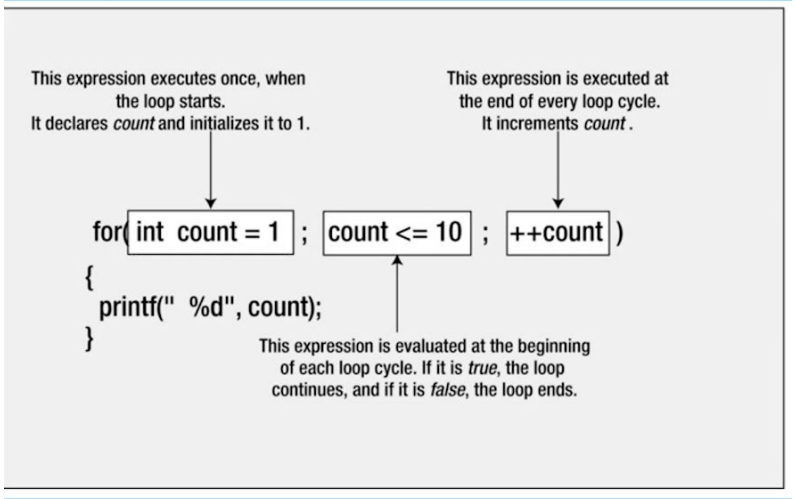
- The **`action_per_iterations`** is executed at the end of each loop iteration.
  - ➔ Usually an increment or decrement of one or more loop control variables.
  - ➔ Can modify several variables here, just need to use commas

EXAMPLE:

```
for(int i = 1, j = 2 ; i <= 5 ; ++i, j = j +2)
printf(" %5d", i*j);
```

- The output produced by this fragment will be the values 2, 8, 18, 32 and 50 on a single line
- We doing three separate things on a single line of code.
  1. Initializing loop control variables and any other variables
  2. We having our boolean expression after our first semicolon our exit condition
  3. Modifying the control variable or any other variable or any variable we want as the last part of the for statement.

For syntax



Example:

```
Unsigned long long sum = 0;     //Stores the sum of the integers
```

```
Unsigned int count = 0;          //The number of integers to be summed

// Red the number of integers to be summed
printf("\nEnter the number of integers you want to sum:');
scanf("%u", &count);

//Sum integers from 1 to count
for (unsigned int i = 1; i <= count; ++i)
sum +=i;

//Or
for(unsigned int i = 1; i <= count; sum += ++i);

printf("\nTotal of the first %u numbers is %llu\n", count, sum);
```

Infinite Loop

- You have no obligation to put any parameters in the for loop statement.
  for(;;)
  {
  /* statements */
  }

- The condition for continuing the loop is absent , the loop will continue indefinitely
  - ➔ Sometimes useful for monitoring data or listening for connections,