

Looping statements in C for, while and do while

DSA-I

2023

SY(COMP)

COEP

(Shrida Kalamkar)

Control Statements

```
graph TD; A[Control Statements] --> B[Conditional Statements]; A --> C[Looping Statements]; A --> D[Jumping Statements]; B --> B1["if statement"]; B --> B2["if else statement"]; B --> B3["nested if else"]; B --> B4["else if construct"]; B --> B5["switch statement"]; C --> C1["for loop"]; C --> C2["while loop"]; C --> C3["do while loop"]; D --> D1["break statement"]; D --> D2["continue statement"]; D --> D3["goto statement"];
```

Conditional Statements

if statement
if else statement
nested if else
else if construct
switch statement

Looping Statements

for loop
while loop
do while loop

Jumping Statements

break statement
continue statement
goto statement

The Loop Control Structure

- ❖ So far we have used either a sequential or a decision control instruction. (if –else, switch – case)
- ❖ the executions were carried out in a **fixed order**
- ❖ an appropriate set of instructions were executed depending upon the outcome of the **condition being tested Exactly once**
- ❖ 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.

The Loop Control Structure

- ❖ Every programming language has the feature to instruct to do such repetitive tasks with the help of certain form of statements.
- ❖ The process of repeatedly executing a collection of statement is called **looping**.
- ❖ The statements gets executed many number of times based on the condition.
- ❖ But if the condition is given in such a logic that the repetition continues any number of times with no fixed condition to stop looping those statements, then this type of looping is called **infinite looping**.

The Loop Control Structure

- ❖ A loop statement allows us to execute a statement or group of statements multiple times.

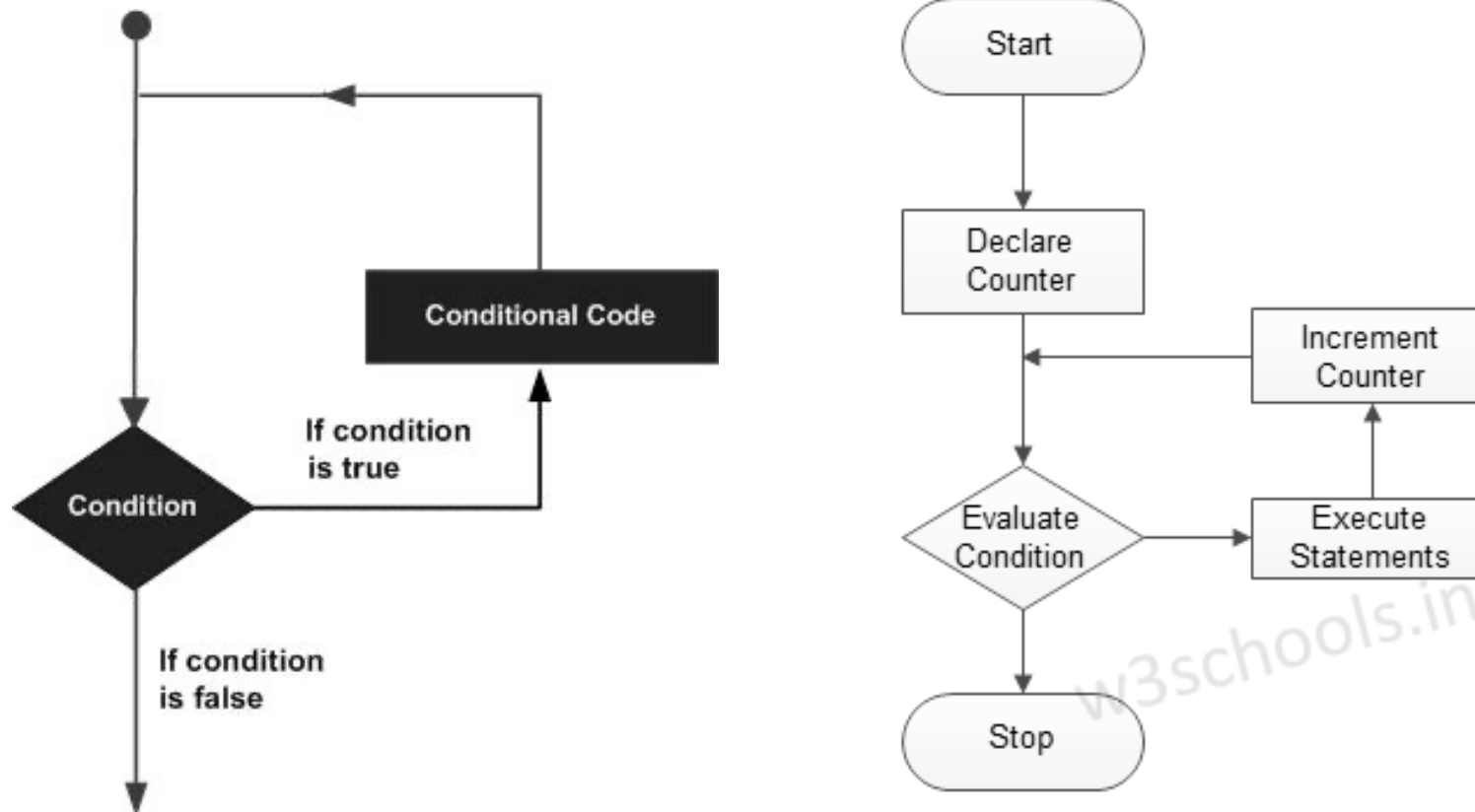
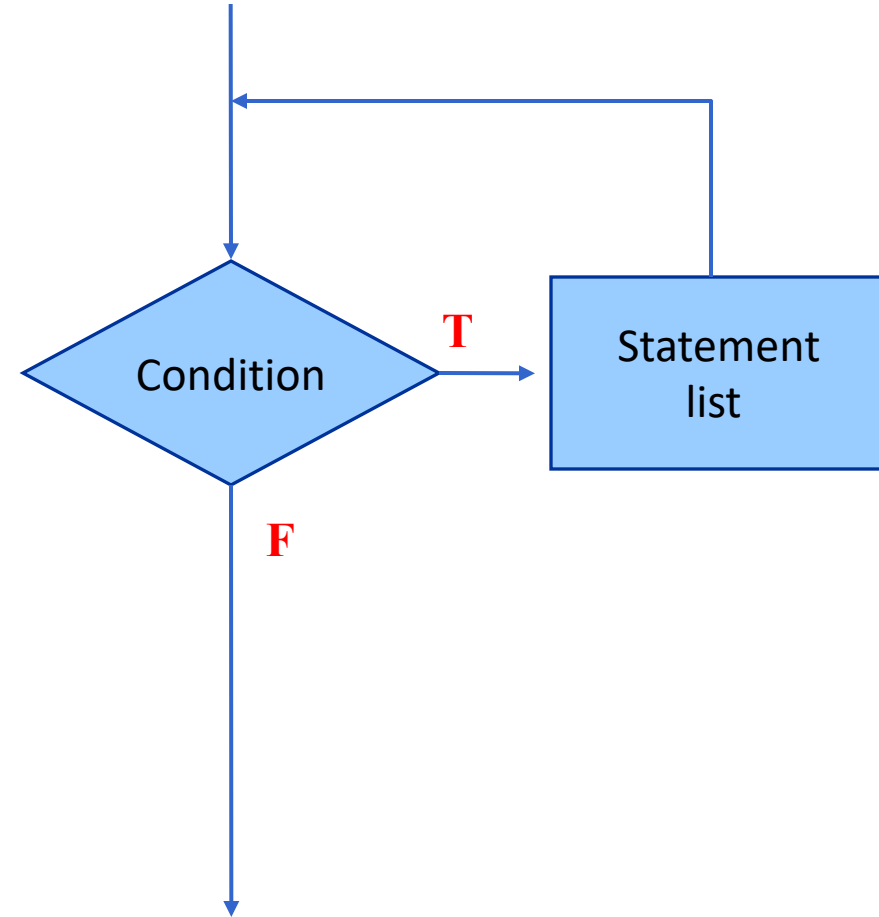
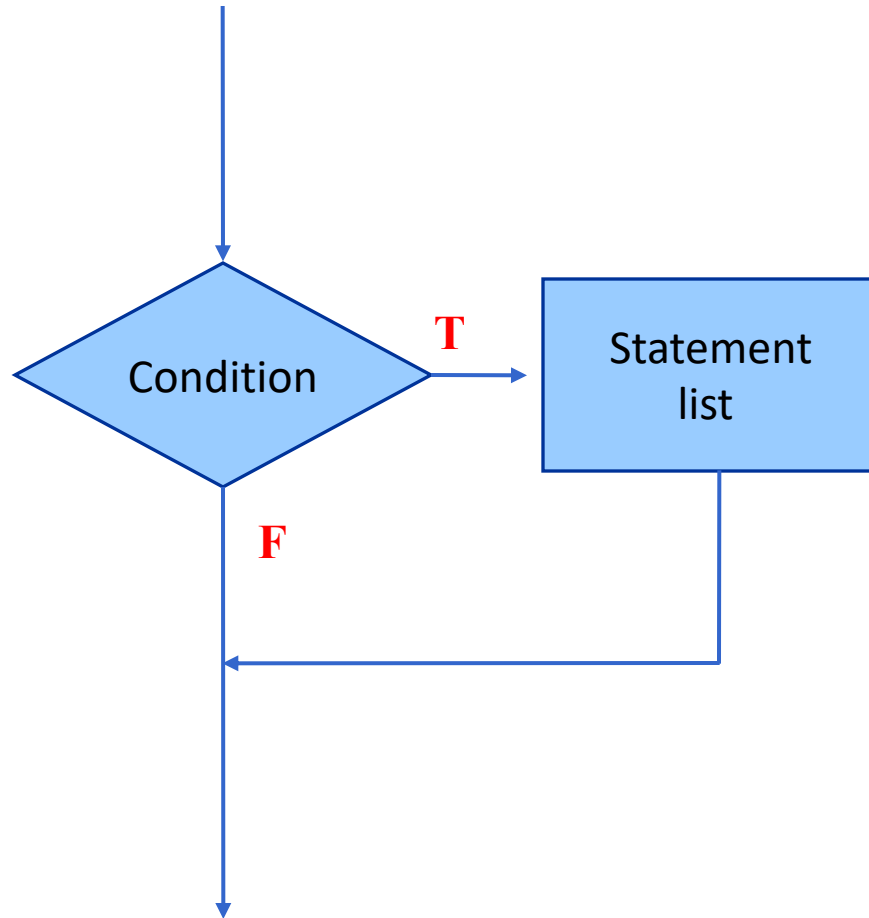


Figure – Flowchart of Looping

Branching

vs

Loops



Types of Loop

- ❖ There are three methods by way of which we can repeat a part of a program. They are:
 - (a) Using a **for statement**
 - (b) Using a **while statement**
 - (c) Using a **do-while statement**
- ❖ It is often the case in programming that you want to do something a fixed number of times
 - Ex. calculate gross salaries of ten different persons,
 - Ex. Convert temperatures from centigrade to Fahrenheit for 15 different cities.

The for loop

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 format for a for loop

```
for(Initialization_action; Condition; Condition_update) {  
    statement_list;  
}
```


The for Loop

The syntax of a for loop in C programming language is –

```
for ( init; condition; increment ) {  
    statement(s);  
}
```

The for Loop : flow of control

- 1 . The init step is executed first, and only once.

This step allows you to declare and initialize any loop control variables.

2. Next, the condition is evaluated.

If it is true, the body of the loop is executed.

If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.

3. After the body of the 'for' loop executes, the flow of control jumps back up to the increment/modication statement.

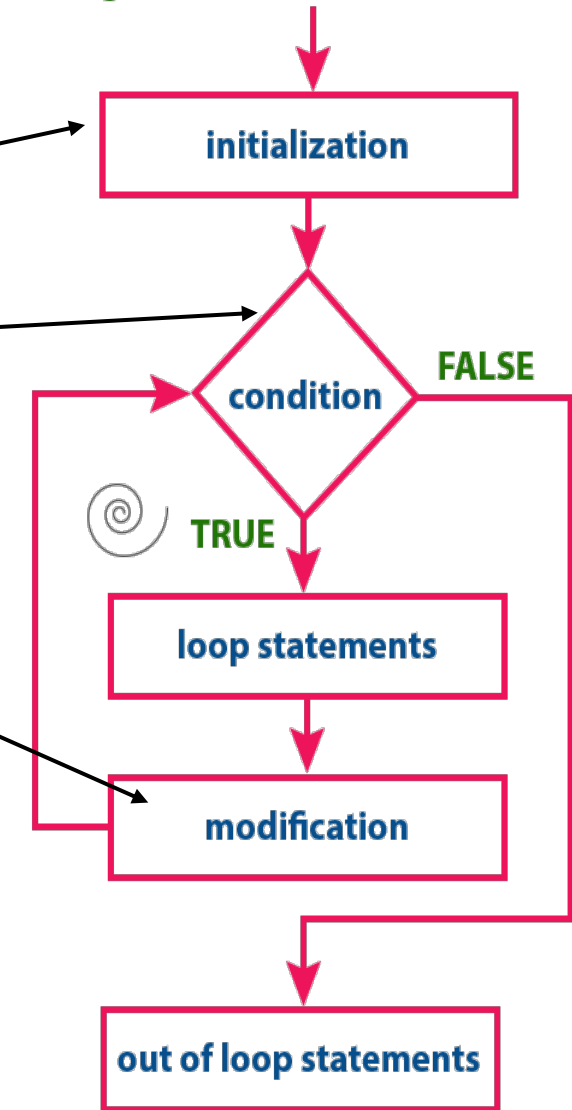
This statement allows you to update any loop control variables.

4. The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Working of for loop in C

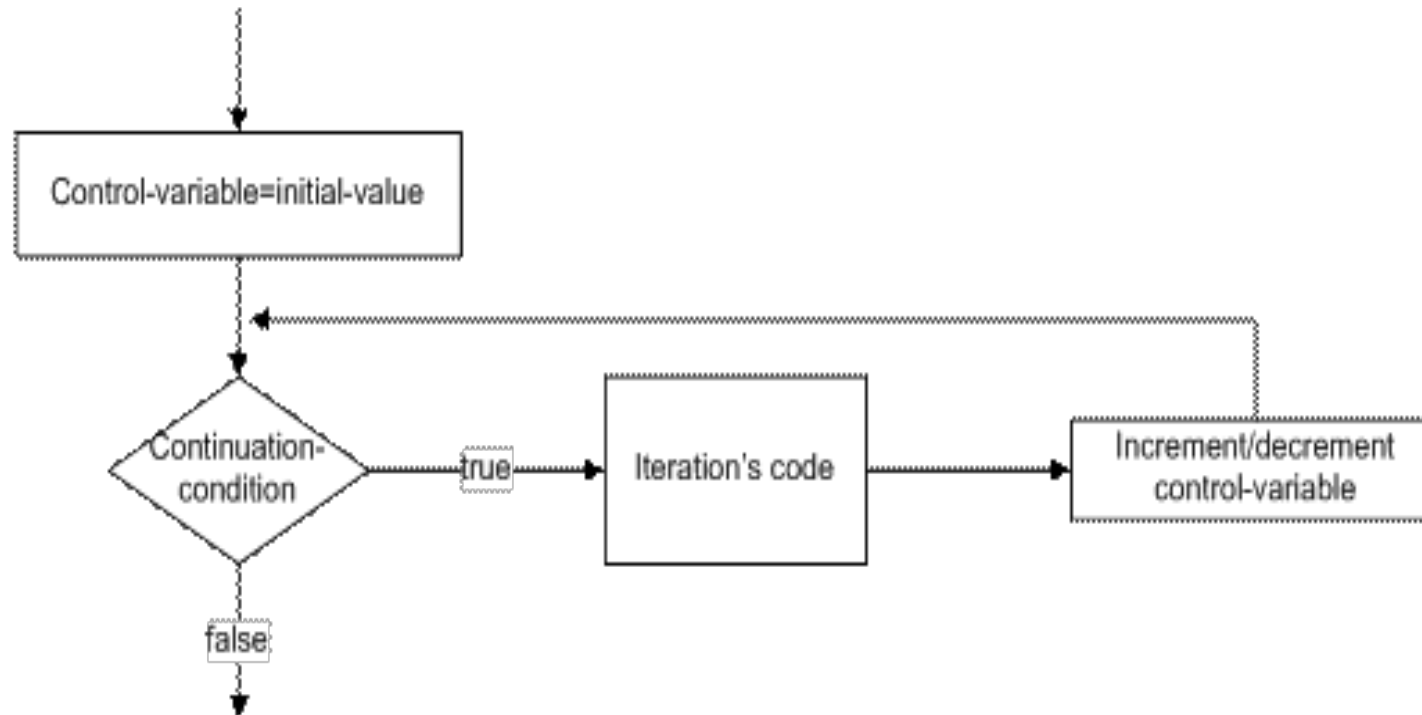
1	int main() {
2	int i, sum = 0;
3	for(i = 0; i < 11; i++) {
4	printf("%d", i);
5	sum = sum + i;
6	}
7	printf("%d", sum);
8	return 0;
9	}

Execution flow diagram:



The for loop

```
for(control-variable initialization; continuation condition; increment/decrement-  
control variable ) {  
    // body of loop  
}
```



Code to display integers from 1 -10

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

```
int main() {
```

```
    int i;
```

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

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

```
}
```

Note: The curly brackets { }, can be skipped if the for loop has only 1 statement.

If the brackets are missing then ONLY very next statement will be iterated

Try following now:


- 1 Print numbers from 10 to 1
- 2 Print numbers from n to m. n and m are accepted from user.
- 3 Print all numbers divisible by 'n' , between 1 to 1000. 'n' accepted from user
- 4 Print sum of numbers from 1 to 10
5. Print Alphabets from A – Z, a- z, z-a, Z-A

Print alphabets from 'A' – 'Z'

```
include<stdio.h>
int main() {
    char i;
    for(i = 'A'; i <= 'Z'; i++) {
        printf("%c\n", i);
    }
    return 0;
}
```

What will be the output?

```
include<stdio.h>
int main() {
    char i;
    for(i = 'A'; i <= 'Z'; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```



OUTPUT?
NOTE THE CHANGE

Example 1: For – Simple interest

```
/* Calculation of simple interest for 3 sets of p, n and r */  
void main ( ) {  
    int p, t, count ;  
    float r, si ;  
    for ( count = 1 ; count <= 3 ; count = count + 1 ) {  
        printf ( "Enter values of p, t, and r " ) ;  
        scanf ( "%d %d %f", &p, &t, &r ) ;  
        si = p * t * r / 100 ;  
        printf ( "Simple Interest = Rs.%f\n", si ) ;  
    }  
}
```

Example 2: For - Factorial

Factorial of n is $n*(n-1)*(n-2)*...2*1$

```
int i, n, f=1;
printf("Enter a number to find factorial");
scanf("%d",&n);
for (i=1; i<=n; ++i) {
    f = f* i;
}
printf("The factorial of %d is %d ",n,f);
```

Additional features of for loop

■ The comma operator

We can give several statements separated by commas in place of “Initialization”, “condition”, and “updating”.

Example

```
for (fact=1, i=1; i<=10; i++)  
    fact = fact * i;
```

```
for (sum=0, i=1; i<=N; ++i)  
    sum = sum + i * i;
```

- We can **use expression** in assignment statements of initialization and update section

Example:

```
for( x = ( m + n ) / 2 ; x > 0 ; x = x/2 )
```

- We can **omit one or more sections**

Example:

```
int m=5;  
for ( ; m != 100 ; ) {  
    statements;  
    m = m + 5 ;  
}
```

for(expr1; expr2; expr3)

Most commonly, expr1 and expr3 are assignments or function calls and expr2 is a relational expression.

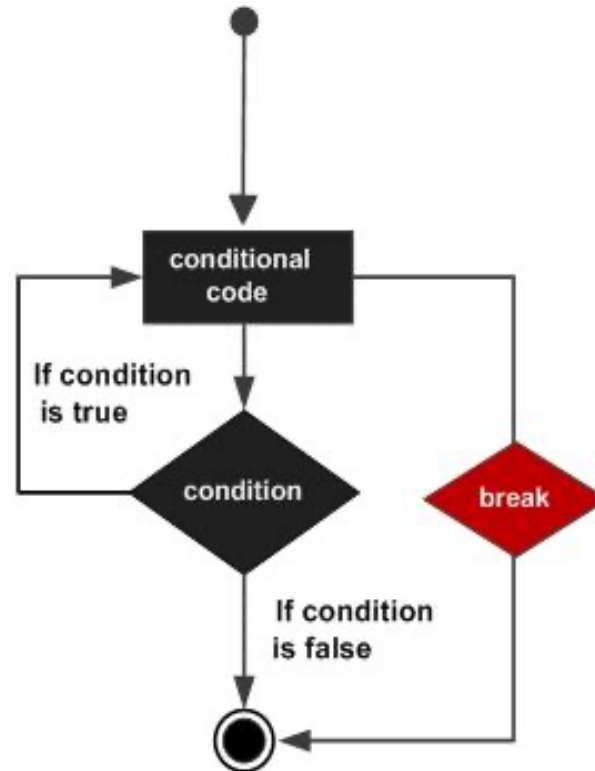
Any of the three parts can be omitted, although the semicolons must remain.

If expr1 or expr3 is omitted, it is simply dropped from the expansion.

If the test, expr2, is not present, it is taken as permanently true

Break Statement in C

- Break statement are used for terminates any type of loop e.g, while loop, do while loop or for loop.
- The break statement terminates the loop body immediately and passes control to the next statement after the loop.
- In case of inner loops, it terminates the control of inner loop only.



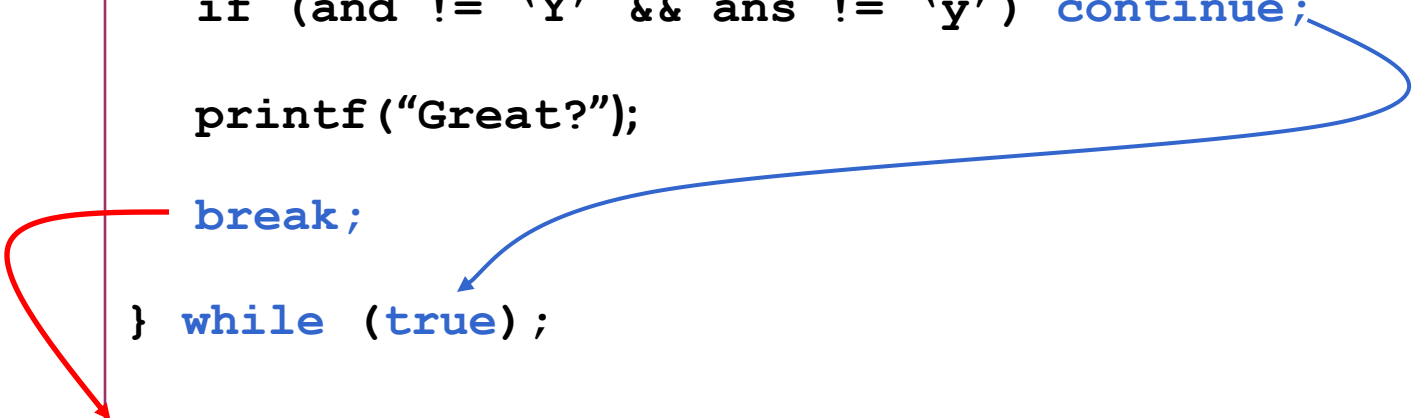
Continue Statement in C

- The continue statement in C programming works somewhat like the break statement.
- Instead of forcing termination, it forces the **next iteration** of the loop to take place, skipping any code in between.
- In a **for loop**, it will force the program to update (increment or decrement) the loop variable and then check the condition immediately.
- For the **while** and **do...while** loops, continue statement causes the program control to pass to the conditional tests.

Continue in a loop

The **continue** statement in a loop will force the program to **check** the loop condition immediately.

```
do {  
    printf("Will you pass this exam?");  
    scanf("%c",&ans);  
  
    if (and != 'Y' && ans != 'y') continue;  
  
    printf("Great?");  
  
    break;  
} while (true);  
....
```



break/ continue

```
int i = 0;
for(;;) {
    if(i > 10)
        break;
    else
        continue;
    printf("%d", i);
    i = i + 1;
}
```

```
int i = 0;
for(; i <= 10;) {
    printf("%d", i);
    i = i + 1;
}
```

```
int i = 0;
for(; ; i = i + 1) {
    if(i > 10)
        break;
    else
        continue;
    printf("%d", i);
}
```

Write a code to keep accepting numbers and adding them till the sum exceeds 100

```
include<stdio.h>
int main() {
    int i, sum,n;
    for(i = 1, sum = 0; ; i++){
        scanf("%d", &n);
        sum += n;
        if(sum <= 100)
            continue;
        else
            break;
    }
    printf("Total numbers read = %d\nSum = %d", i, sum);
}
```

Nested loops (loop in a loop)

The syntax for a nested for loop statement in C is as follows –

```
for (init; condition; increment) {  
    for (init; condition; increment) {  
        statement(s);  
    }  
    statement(s);  
}
```

Nested loops (loop in a loop)

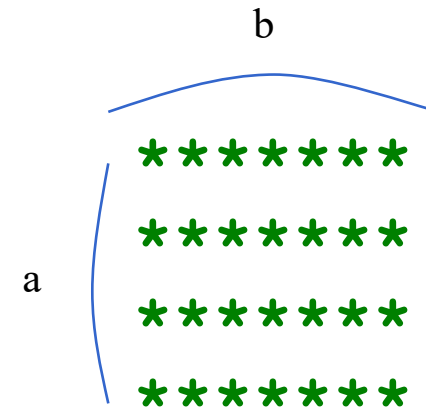
In Nested loop one loop is place within another loop body.

When we need to repeated loop body itself n number of times use nested loops.

Nested loops can be design upto 255 blocks.

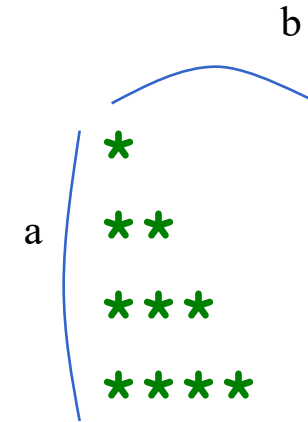
```
printf("%d%d",a,b); //a=4,b=7
```

```
for (i = 0; i < a; i++){  
    for (j=0; j<b; j++) {  
        printf("*");  
    }  
    printf("\n");  
}
```



Nested loops example

```
int a,b;  
Printf("Enter two values");  
scanf("%d%d",&a,&b);  
  
for (i = 0; i < a; i++) {  
    for (j = 0; j < b; j++)  
        if (j > i)  
            break;  
    printf("*");  
}  
printf("\n");  
}
```

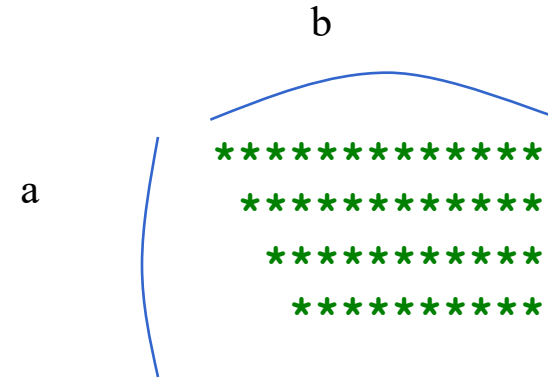


Nested loops example

```
int a,b;

scanf ("%d%d", &a, &b) ;

for (int i = 0; i < a; i++) {
    for (int j=0; j<b; j++) {
        if (j < i)
            printf(" ");
        else
            printf("*");
    }
    printf("\n");
}
```



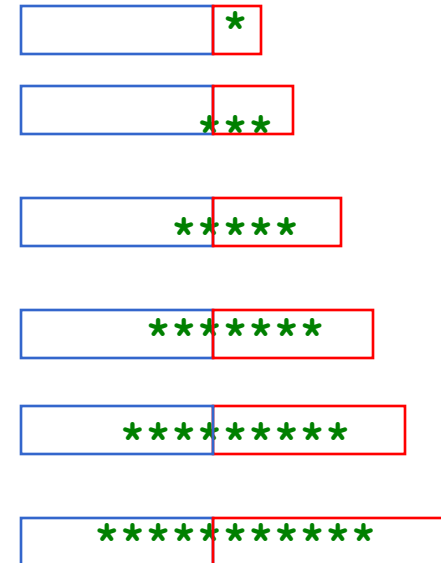
Nested loops example

```
int a,i,j;  
scanf("%d",&a);
```

```
for (i = 0; i < a; i++) {  
    for (j=0; j<a; j++) {  
        if (j < a-i)  
            printf(" ");  
        else printf("*");  
    }  
}
```

```
for (j=0; j<a; j++) {  
    if (j > i)  
        break;  
    printf("*");  
}
```

```
printf("\n");  
}
```



Comparison with Python 'for'

```
#Syntax of for statement in Python
for i in range(start, stop, step):
    statement1
    statement2
```

```
# Example of for loop in Python
for i in range(1, 11, 1):
    print(i)
```

```
/* C Code Segment*/
int start , stop = 11, step = 1;
for(start = 1; start < stop; start = start +1)
    printf("%d", start);
```


Syntax:

```
for(initialization; condition; modification) {
```

block of statements

```
}
```

```
for(expr1; expr2; expr3) {
```

```
}
```

Grammatically, the three components of a for loop are expressions

Empty and infinite loop

```
for (; ; );  
for (i = 0; i < 1000000; i++);
```

```
for (i = 1; i > 0; i++) {  
    loop body  
}
```

Write C Program using for loop:

1. Write a C program to find factorial
2. Write a C Program to reverse a given number. (For e.g., user entered 54823 so its reverse is 32845)
3. Write a C Program to find product of digits of a number. (For e.g., user entered 2534, product of digits is $2 \times 5 \times 3 \times 4 = 120$ and user should not enter 0 as a digit)
4. Sum of series :

$$x + x/1 + x/2! + x/3! \dots x/n!$$

Write Code to get the following output:

```
1          #include<stdio.h>
1 2
1 2 3      int main() {
1 2 3 4      int i,j;
1 2 3 4 5    for(i = 1; i <= 5; i++) {
              for(j = 1; j <= i; j++)
                  printf("%d", j);
              printf("\n");
            }
            return 0;
        }
```

Write Code to get the following output:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Examples:

