

First Year Engineer in computer science

CHAPTER 4:

iteration Statements

(Loops)

Iteration (Loop) statements

2

- There may be a situation, when you need to execute a block of code several number of times.
- A loop statement allows us to execute a statement or group of statements multiple times.

Example: We want to display the multiplication table for the number 5.

$$1 \times 5 = 5$$

$$2 \times 5 = 10$$

$$3 \times 5 = 15$$

....

$$10 \times 5 = 50$$

Iteration (Loop) statements

3

Simple Solution:

```
printf("1 x 5 = %d \n",1*5);  
printf("2 x 5 = %d \n",2*5);  
printf("3 x 5 = %d \n",3*5);  
printf("4 x 5 = %d \n",4*5);  
printf("5 x 5 = %d \n",5*5);  
printf("6 x 5 = %d \n",6*5);  
printf("7 x 5 = %d \n",7*5);  
printf("8 x 5 = %d \n",8*5);  
printf("9 x 5 = %d \n",9*5);  
printf("10 x 5 = %d \n",10*5);
```

Too long solution !
We repeated roughly 10
times the same statement



We can do better using
loops

Iteration (Loop) statements

4

We can see that there is a counter taking values from 1 to 10.

10 times we need to:

1. display the counter value

2. display " x 5 = "

3. display the result of counter * 5

4- add a newline "\n".

printf("1	x 5 = %d \n",	1*5)
printf("2	x 5 = %d \n",	2*5)
printf("3	x 5 = %d \n",	3*5)
printf("4	x 5 = %d \n",	4*5)
printf("5	x 5 = %d \n",	5*5)
printf("6	x 5 = %d \n",	6*5)
printf("7	x 5 = %d \n",	7*5)
printf("8	x 5 = %d \n",	8*5)
printf("9	x 5 = %d \n",	9*5)
printf("10	x 5 = %d \n",	10*5)

Iteration (Loop) statements

5

- To obtain this result we need to repeat the following statement:

`printf("%d x 5 = %d",i, i*5);`

- The counter `i` takes values from 1 to 10.

<code>printf("1</code>	<code>x 5 = %d", 1*5)</code>
<code>printf("2</code>	<code>x 5 = %d", 2*5)</code>
<code>printf("3</code>	<code>x 5 = %d", 3*5)</code>
<code>printf("4</code>	<code>x 5 = %d", 4*5)</code>
<code>printf("5</code>	<code>x 5 = %d", 5*5)</code>
<code>printf("6</code>	<code>x 5 = %d", 6*5)</code>
<code>printf("7</code>	<code>x 5 = %d", 7*5)</code>
<code>printf("8</code>	<code>x 5 = %d", 8*5)</code>
<code>printf("9</code>	<code>x 5 = %d", 9*5)</code>
<code>printf("10</code>	<code>x 5 = %d", 10*5)</code>

Iteration (Loop) statements

6

C programming language provides three types of loops:

1. **For** loop statement
2. **While** loop statement
3. **Do while** loop statement

For loop statement

7

→ Algorithmic (pseudo-code):

```
For counter ← initial value to end value do  
    processing  
endFor
```

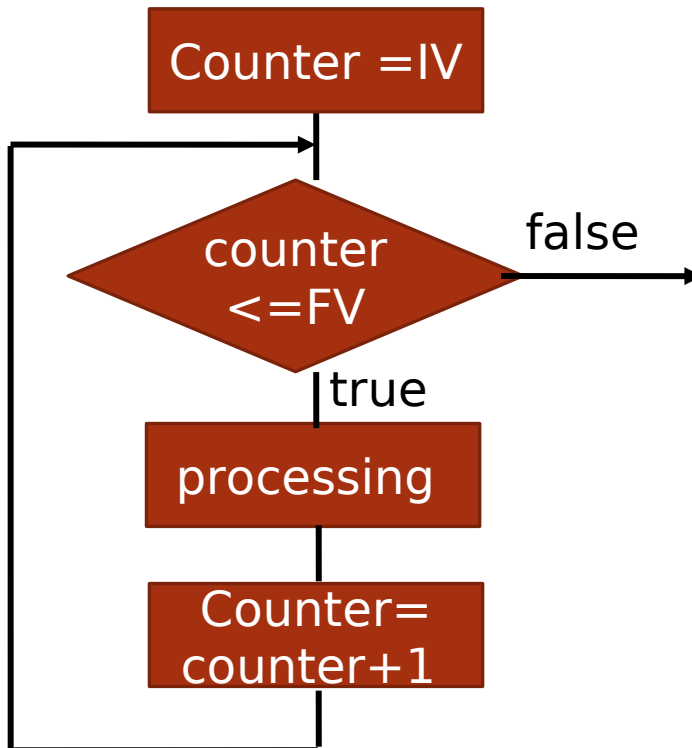
The integer counter variable is initialized to the initial value and it is incremented automatically after each iteration until the final value.

For each value of the counter the statements composing the processing are executed.

For loop statement

8

- Algorithm (flowchart):



IV: Initial Value

FV: Finale Value

For loop statement

9

C syntax:

```
for ( init; condition; increment )  
{  
    //statements ;  
}
```

For loop statement

10

The flow of control in a for loop:

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 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 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.

For loop statement

11

Example: Multiplication table for number 5

Algorithm:

```
begin
for (i ← 1 to 10) do
    display (i, '×5=', i*5)
endfor
end
```

Code C :

```
#include <stdio.h>

int main()
{
    int i;
    for(i = 1; i <= 10; i++)
        printf("%d x 5 = %d \n", i, i*5);
    return 0;
}
```

For loop statement

12

Exercise 1: write a C code to compute the sum:
 $S=1+2+3+\dots+N$. N is a positive integer.

Exercise 2: write a C code to compute the factorial of a positive integer number.

while loop statement

13

Algorithm:

```
while (condition) do  
    processing  
endwhile
```

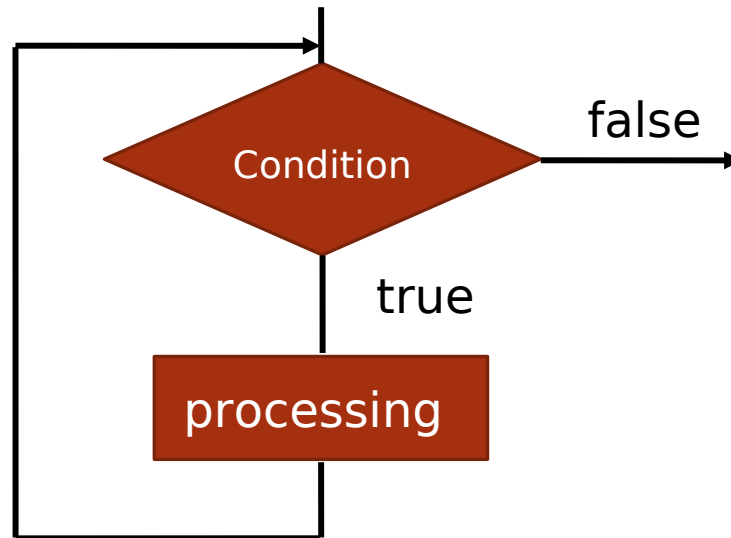
While the condition is evaluated to **true** the bloc of statements (processing) is executed. When the condition becomes **false**, program control passes to the line immediately following the loop.

Remark : key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

while loop statement

14

flowchart:



while loop statement

15

C syntax:

```
while (Condition)
{
    //statements ;
}
```

while loop statement

16

Example : Multiplication table for number 5

```
#include <stdio.h>

int main()
{
    int i= 1;
    while(i<=10)
    {
        printf("%d x 5 = %d \n",i,i*5);
        i++;
    }
    return 0;
}
```


while loop statement

17

Exercise 3 : Write a C program that asks the user to enter a number 'n' between 10 and 20. The program should then calculate the square of this number. If the entered number is not within the specified range, the program should display an error message (number < 10 or number > 20) and prompt the user to enter a new value. The program will continue to request input until a valid number is finally entered.

“Do ... while” statements

18

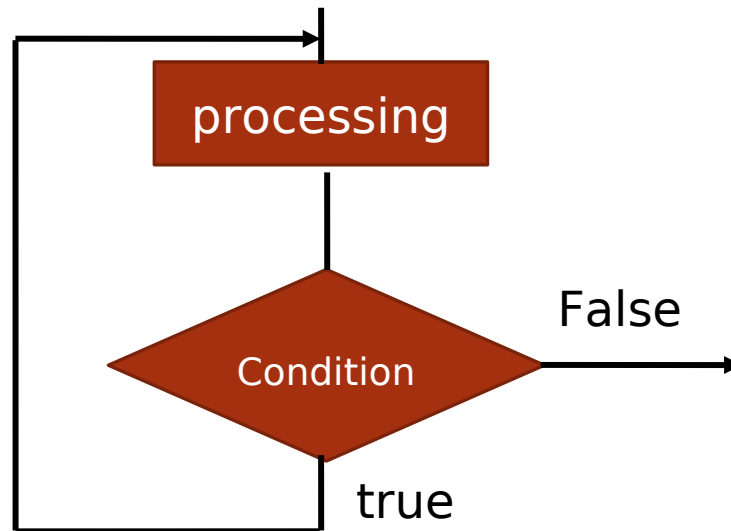
Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.

A **do...while** loop is similar to a **while** loop, except that a **do...while** loop is guaranteed to execute at least one time.

“Do ... while” statements

19

flowchart:



“Do ... while” statements

20

Example : Multiplication table for number 5

```
#include <stdio.h>

int main()
{
    int i = 1;
    do{
        printf("%d x 5 = %d \n",i,i*5);
        i++;
    } while(i<=10);

    return 0;
}
```

Which type of loop to use ?

21

Use the structure that best reflects the idea of the program you want to implement:

- If the block of statements should not be executed if the condition is false, then use **while** or **for**.
- If the block of statements must be executed at least once, then use **do-while**.
- If the number of executions of the block of statements is known in advance, then use **for**.
- If the block of statements must be executed as long as an external condition is true, then use **while**.
- The choice between **for** and **while** is often just a matter of preference or habit.

nested loops in C

22

C programming language allows to use one loop inside another loop.

Example of nested loops:

```
int i ;
int j =1;
for ( i = 1 ; i <= 3 ; i ++ )
{
    j =1 ;
    while (j <= 4)
    {
        printf (" i =% d et j =% d !\n ", i , j);
        j++;
    }
}
```

nested loops in C

23

C programming language allows to use one loop inside another loop.

Example of nested loops:

```
int i ;
int j =1;
for ( i = 1 ; i <= 3 ; i ++ )
{
    j =1 ;
    while (j <= 4)
    {
        printf (" i =% d et j =% d !\n ", i , j);
        j++;
    }
}
```

Output

```
i = 1 et j = 1
i = 1 et j = 2
i = 1 et j = 3
i = 1 et j = 4
-----
i = 2 et j = 1
i = 2 et j = 2
i = 2 et j = 3
i = 2 et j = 4
-----
i = 3 et j = 1
i = 3 et j = 2
i = 3 et j = 3
i = 3 et j = 4
-----
```

nested loops in C

24

Exercise 4: Write a C program that asks the user to enter a positive integer 'n' and displays all numbers from 0 to n. If the entered number is not a positive integer, the program should display an error message (n is not positive) and prompt the user to enter a new value. The program will continue to request input until a valid number is finally entered.

break statement in C

25

- When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

Syntax of the **break** statement in a **for** loop:

```
for ( init ; Condition ; increment )  
{  
    // statements to loop  
    if condition  
        break;  
    // more statements to loop ;  
}  
→ // statements outside the loop ;
```

break statement in C

26

→ Syntax of the **break** statement in a **while** loop:

while condition

{

 // statements to loop ;

if condition

break;

 // more statements to loop:

}

// statements outside the loop ;



break statement in C

27

→ Syntax of the **break** statement in a **do ...while** loop:

do
{

// more statements to loop;

if condition

break;

// more statements to loop;

} **while** condition;

// statements outside the loop ;



break statement in C

28

Example : **break** statement in a **for** loop

```
for ( i = 1 ; i <= 11 ; i++)  
{  
    if ( i >= 6 )  
        break;  
    printf( " %d \t ", i);  
}  
printf("The end !");
```

break statement in C

29

Example : **break** statement in a **for** loop

```
for ( i = 1 ; i <= 11 ; i++)  
{  
    if ( i >= 6 )  
        break;  
    printf( " %d \t ", i);  
}  
printf("The end !");
```

Execution



1 , 2 , 3 , 4 , 5
The end !

Get out from the **for**
loop without executing
next statements

continue statement in C

30

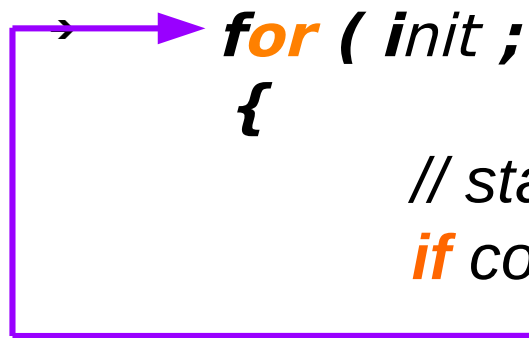
The **continue** statement in C programming language forces the next iteration of the loop to take place, skipping any code in between.

For the **for** loop, continue statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, continue statement causes the program control passes to the conditional test.

continue statement in C

31

Syntax of the **continue** statement in the **for** loop:

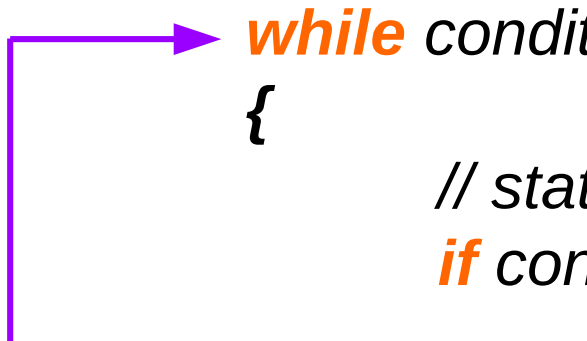


```
for ( init ; Condition ; increment )  
  {  
    // statements to loop  
    if condition  
      continue;  
    / more statements to loop  
  }  
  // statements outside to loop
```

continue statement in C

32

Syntax of the **continue** statement in the **while** loop:

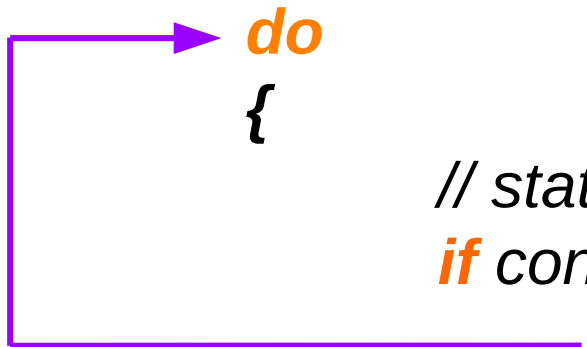


```
while condition
{
    // statements to loop
    if condition
        continue;
    // more statements to loop
}
// statements outside to loop
```


continue statement in C

33

Syntax of the **continue** statement in the **do while** loop:



```
do  
{  
    // statements to loop  
    if condition  
        continue;  
    // more statements to loop  
} while condition  
// statements outside to loop
```

continue statement in C

34

Example : the **continue** statement in a **for** loop:

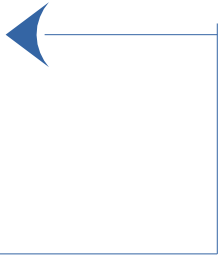
```
for ( i = 1 ; i <= 6 ; i++)  
{  
    if ( i == 4 )  
        continue;  
    printf( "the value of i is : %d \n ", i );  
}  
printf("The end !");
```

continue statement in C

35

Example : the **continue** statement in a **for** loop:

```
for ( i = 1 ; i <= 6 ; i++)  
{  
    if ( i == 4 )  
        continue;  
    printf( "the value of i is : %d \n ", i );  
}  
printf("The end !");
```



Go back to **for**
without executing
the next statements

Output

```
the value of i is : 1  
the value of i is : 2  
the value of i is : 3  
the value of i is : 5  
the value of i is : 6  
The end !
```

continue statement in C

36

Example : the **continue** statement in a **while** loop:

```
i = 1;
while (i < 20)
{
    if ( i % 3 == 0 )
    {
        i += 4 ;
        printf(" i = %d " , i);
        continue;
    }
    print("The variable i = %d", i ) ;
    i++;
}
```

continue statement in C

37

Example : the **continue** statement in a **while** loop:

```
i = 1;
while (i < 20)
{
    if ( i % 3 == 0 )
    {
        i += 4 ;
        printf(" i = %d " , i);
        continue;
    }
    print("The variable i = %d", i ) ;
    i++;
}
```

Output

The variable i = 1
The variable i = 2
i= 7
The variable i = 7
The variable i = 8
i= 13
The variable i = 13
The variable i = 14
i= 19
The variable i = 19

The Infinite Loop

38

A loop becomes **infinite loop** if a condition **never becomes false**.

- The **for loop** is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the `for(;;)` construct to signify an infinite loop.

- We can also use **while** or **do while** for infinite loop: **while(1)** which means the condition is always true.
- To get out from an infinite loop you can use the `break` statement.

The Infinite Loop

39

```
#include <stdio.h>

int main ()
{
    for( ; ; )
    {
        printf("This loop will run forever.\n");
    }

    return 0;
}
```

NOTE: You can terminate an infinite loop by pressing **Ctrl + C** keys.