

# Ch.7 Loops

# What you will learn in this chapter



- Understanding the concept of repetition
- while loop
- do-while loop
- for loop
- break and continue statements

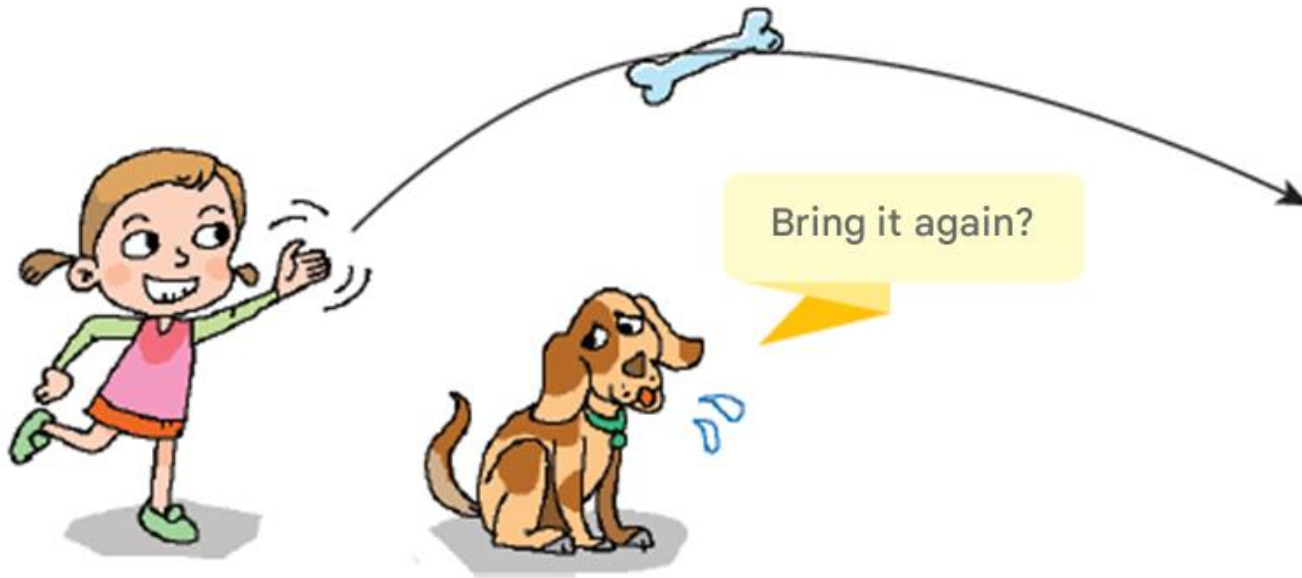
The repetition structure allows you to repeat a series of processes.

First, understand the concept of repetition and learn about the three repetition structures provided in C.



# repeat

- Humans hate repetition, but repetitive tasks are essential in programs.
- Iteration is repeating the same process multiple times .



# Why is repetition necessary ?

Q) Why is a repeating structure necessary ?

A) Because it is necessary to repeat the same processing . To calculate the average grade of 30 students, the same process must be repeated 30 times .



# Why is repetition important ?

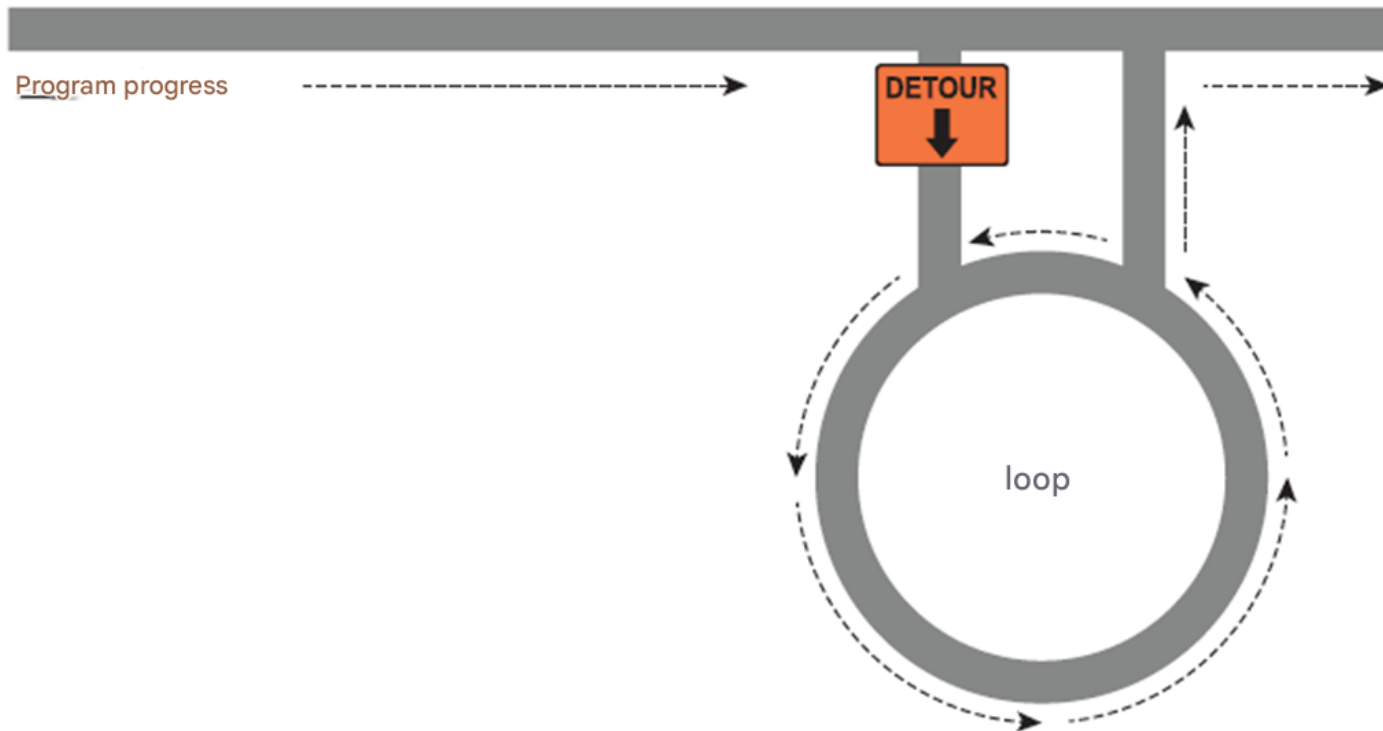
```
printf ( "Hello World! \n" );  
printf ( "Hello World! \n" );  
printf ( "Hello World! \n" );  
printf ( "Hello World! \n" );  
printf ( "Hello World! \n" );
```



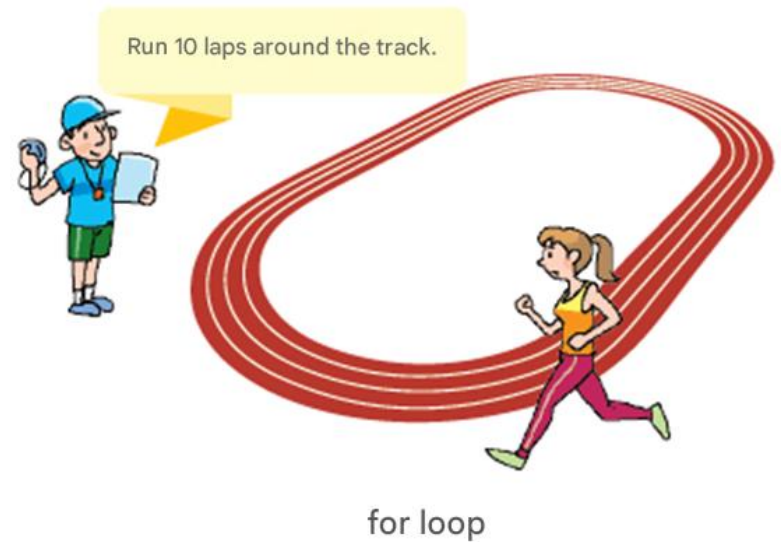
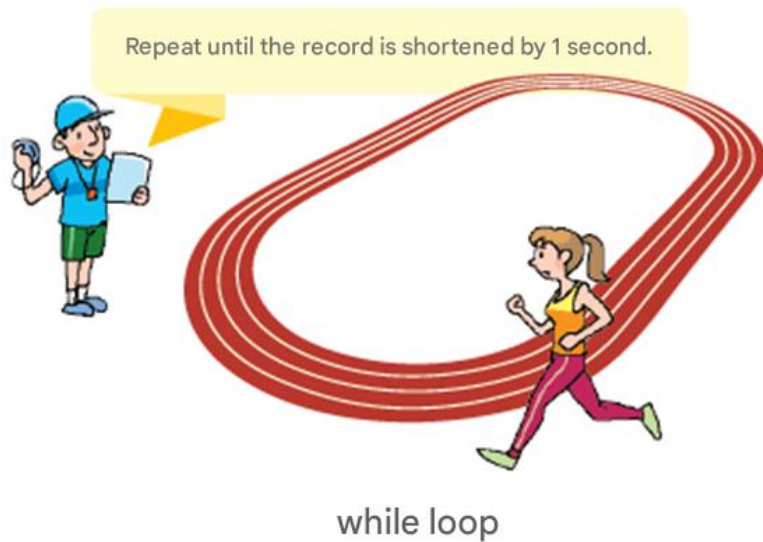
```
for (i = 0; i < 5; i++)  
    printf ( "Hello World! \n" );
```

# repeat structure

- A structure that loops until a certain condition is satisfied.

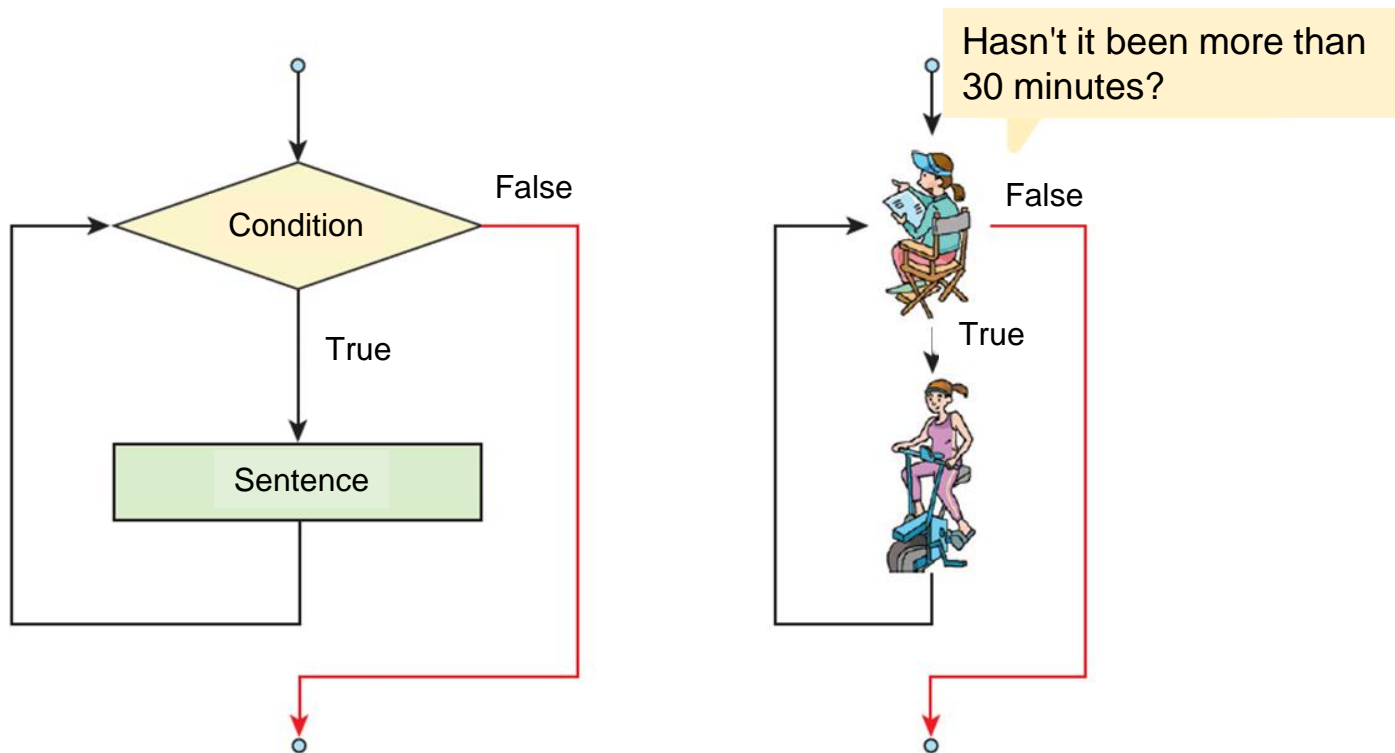


# Types of loops



# while statement

- Repeats execution of statements while a given condition is satisfied.





# while statement

## Syntax

while statement

yes

```
while( i < 10 ) {  
    printf("Hello World!\n");  
    i++;  
}
```

Conditional statement

If the condition is true, the statement is executed repeatedly.

# Example

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    while( i < 5 )
    {
        printf("Hello World! \n");
        i++;
    }

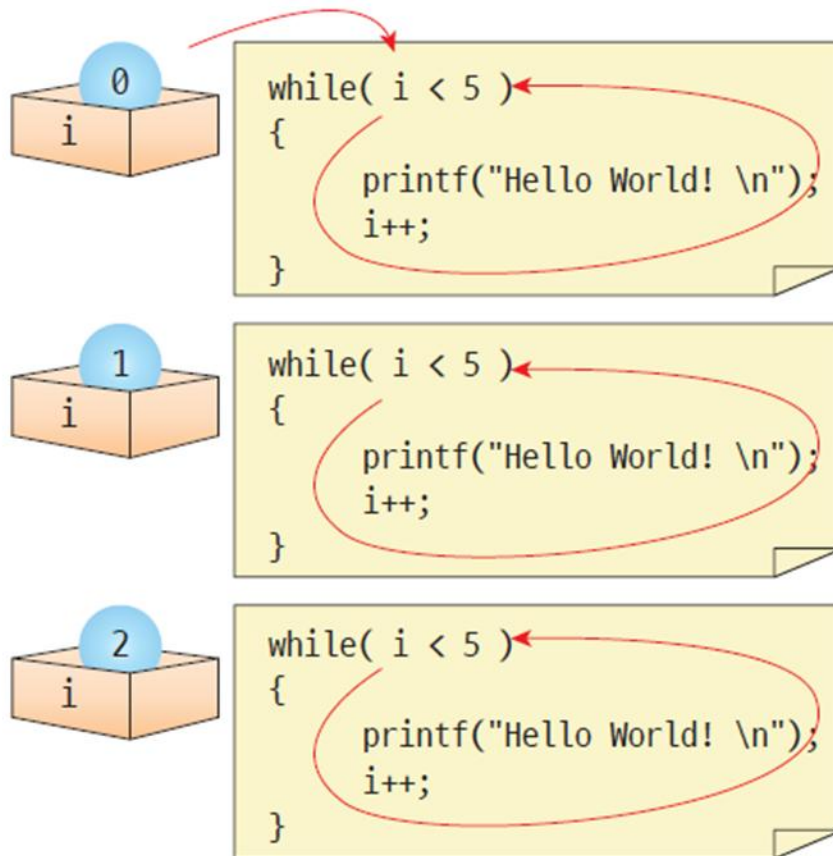
    return 0;
}
```

Repeat condition

Repeat content

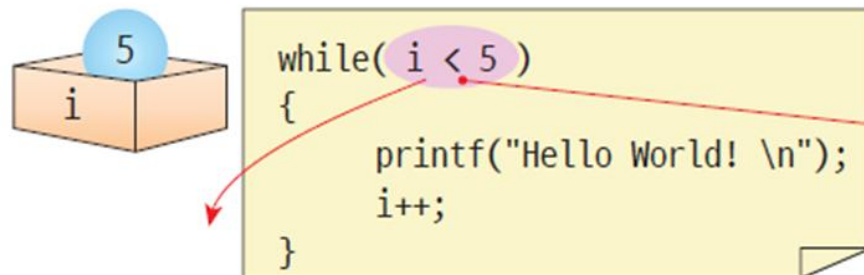
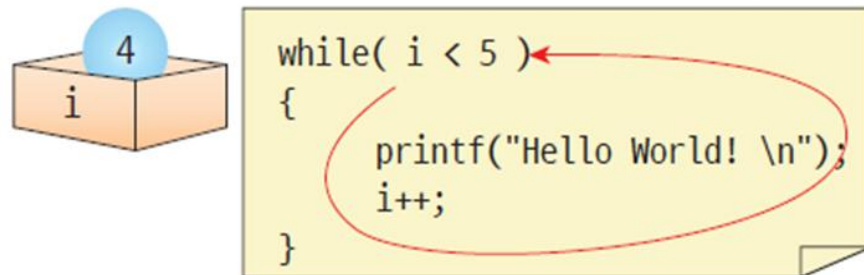
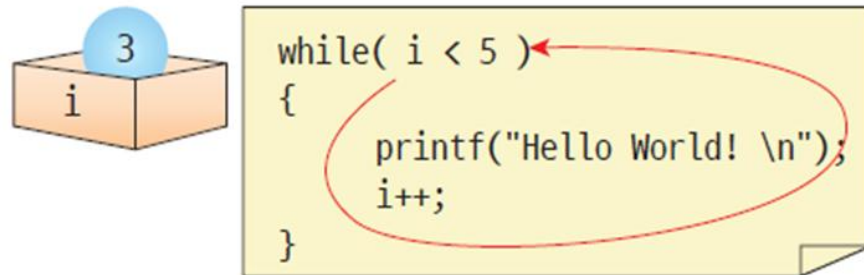
Hello World!  
Hello World!  
Hello World!  
Hello World!  
Hello World!

# while statement execution process



Number of repetitions	value of	(i<5)	Repeat or not
#1	0	True	repeat
#2	1	True	repeat
#3	2	True	repeat
#4	3	True	repeat
#5	4	True	repeat
#6	5	False	stop

# while statement execution process



Number of repetitions	value of i	(i<5)	Repeat or not
#1	0	True	repeat
#2	1	True	repeat
#3	2	True	repeat
#4	3	True	repeat
#5	4	True	repeat
#6	5	False	stop

The loop stops when the condition becomes false.

# Example #1

```
// Program to print multiplication tables using while loop
#include <stdio.h>

int main( void )
{
    int n;
    int i = 1;

    printf ( " The string you want to print : " );
    scanf ( "%d" , &n);

    while ( i <= 9)
    {
        printf ( "%d*%d = %d \n" , n, i , n* i );
        i ++;
    }

    return 0;
}
```

Enter the number you want to print : 9

9\*1 = 9

9\*2 = 18

9\*3 = 27

....

9\*9 = 81

# Example #2

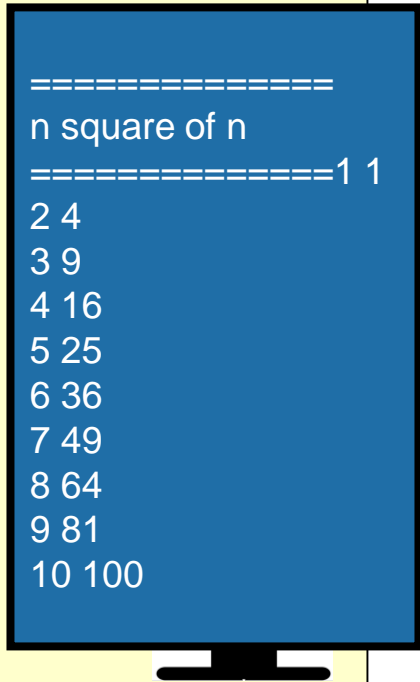
```
// Program to output square values using a while loop
#include <stdio.h>

int main( void )
{
    int n;

    printf( "=====\n");
    printf( " n n squared \n");
    printf( "=====\n");

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

    return 0;
}
```



```
=====
n square of n
=====1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
```

# Example #3

Practice

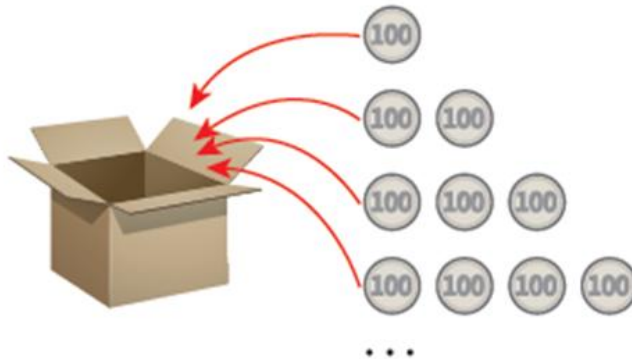
- Program to calculate the sum from 1 to n

Enter an integer : 3  
The sum of 1 to 3 is

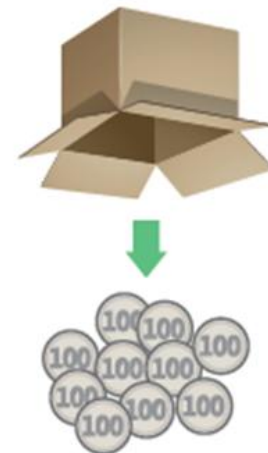
① Prepare an empty container.



② Put numbers 1 to n into the container.



③ Print the number of coins in the container.



# Example #3

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

```
int main( void )
```

```
{
```

```
    int i, n, sum; // variable declaration
```

```
    printf( " Enter an integer :" ); // Output input guidance message
```

```
    scanf( "%d" , &n); // Input integer value
```

```
    i = 1; // initialize variable
```

```
    sum = 0;
```

```
    while (i <= n)
```

```
{
```

```
        sum += i; // Same as sum = sum + i ;
```

```
        i++; // Same as i = i + 1 .
```

```
}
```

```
    printf( " The sum from 1 to %d is %d \n" , n, sum);
```

```
    return 0;
```

```
}
```

Enter an integer : 3  
The sum of 1 to 3 is



# Example #5



- the five values entered by the user and print the result.  
(Repeat input)

```
Enter a value : 10
Enter a value : 20
Enter a value : 30
Enter a value : 40
Enter a value : 50
The total is 150 .
```

# Example #5

```
// Sum program using while loop
#include <stdio.h>

int main( void )
{
    int i, n, sum;

    i = 0; // initialize variables
    sum = 0; // Initialize variables
    while (i < 5)
    {
        printf( " Enter a value : " );
        scanf( "%d" , &n);
        sum = sum + n; // Same as sum += n ;
        i++;
    }
    printf( " The total is %d .\n" , sum);

    return 0;
}
```

Enter a value : 10  
Enter a value : 20  
Enter a value : 30  
Enter a value : 40  
Enter a value : 50  
The total is 150 .

# if and while statements

```
if ( condition )  
{  
...  
...  
}
```

It is executed  
**only once**  
when the  
condition is  
met.

```
while ( condition )  
{  
...  
...  
}
```

If the condition is  
met, it is executed  
**repeatedly multiple**  
**times.**

# Causion: while loops

```
int i = 1;
while (i < 10)
{
    printf ( " Looping \n" );
    i--;
}
```

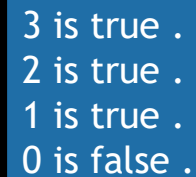
*The variable decreases,  
not increases*

```
int i = 0;
while ( i < 3)
    printf ( " Looping \n" );
    i ++;
```

*Not included in a  
repeating loop .*

# True and false

```
#include <stdio.h>
int main( void )
{
    int i = 3;
    while (i)
    {
        printf ( "%d is true ." , i);
        i--;
    }
    printf ( "%d is false ." , i);
}
```



3 is true .  
2 is true .  
1 is true .  
0 is false .

# conventional form

```
while(i != 0)
{
    ...
}
```



```
while( i )
{
    ...
}
```

# caution



beware of errors

If you use a semicolon(;) at the end of the while condition, only NULL statements are repeated. A statement that only contains a semicolon is called a NULL statement

```
while (i<10) ;
```

It is treated as one sentence and only this is repeated.

```
i++;
```

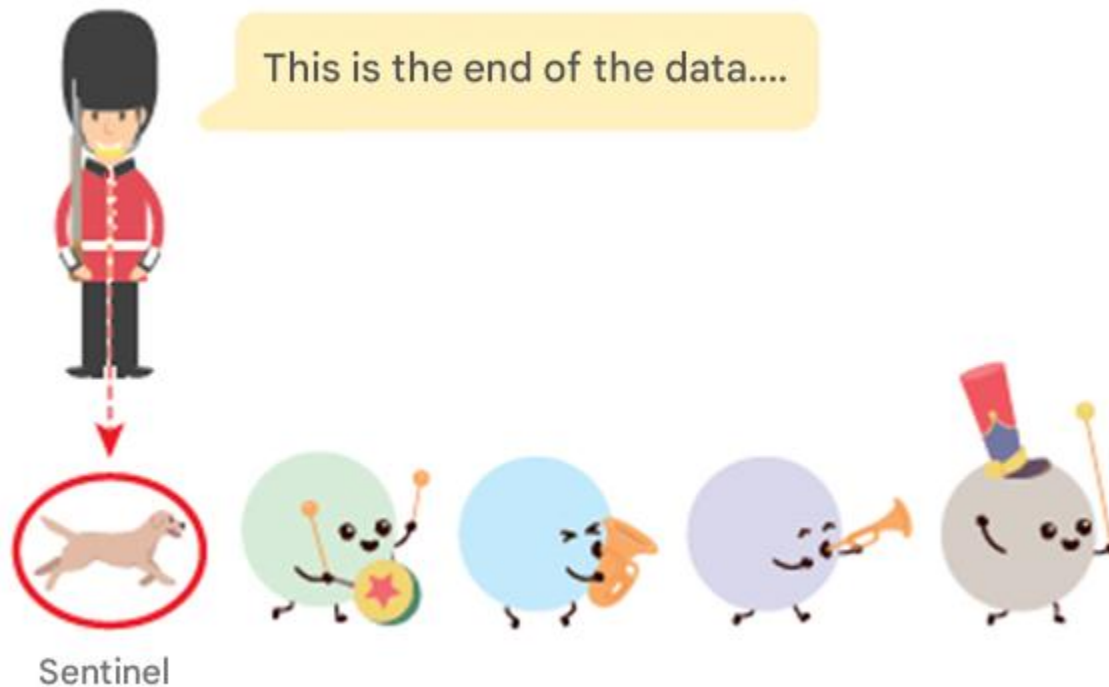
It doesn't repeat.

```
while(i = 2)  
{  
    ...  
}
```

Since the value of the expression is 2, it is always true and thus causes an infinite loop.

# Sentinel ( Using sentinel value )

- Sentinel : A special value that signals the end of input data.





# Problem of calculating the average of grades

- Let's write a program that receives an arbitrary number of grades from the user, calculates their average, and then prints them .
- -1 is used as a sentinel value .

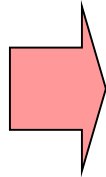
```
Enter  
Enter your grade : 10  
Enter your grade : 20  
Enter your grade : 30  
Enter your grade : 40  
Enter your grade : 50  
Enter your grade : -1  
The average score is 30.000000 .
```



Practice

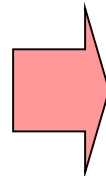
# The problem of finding the average of grades

Find the average of the grades .



1. Initialize the required variables .
2. Enter the grades, calculate the total, and count the number of grades.
3. Calculate the average and display it on the screen .

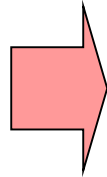
1. Initialize the required variables .



- (1) Initialize sum to 0 .
- (2) Initialize n to 0 .
- (3) Initialize grade to 0 .

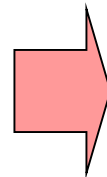
# The problem of finding the average of grades

2. Enter the grades, calculate the total, and count the number of grades .



while if the score is not less than 0  
(1) Read the grade from the user and save it in grade.  
(2) Accumulate this score in sum.  
(3) n is increased by one.

3. Calculate the average and display it on the screen .



(1) Divide sum by n and store it in average.  
(2) Print the average on the screen.

# Sentinel Example 1/2

```
// Program to find the average grade using a while loop
#include <stdio.h>

int main( void )
{
    int grade, n;
    float sum, average;

    // Initialize the necessary variables .
    n = 0;
    sum = 0;
    grade = 0;

    printf ( " Enter a negative number to end grade entry \n" );
```

# Sentinel Example 2/2

```
// Enter the grades , calculate the total, and count the number of students .
while (grade >= 0)
{
    printf ( " Enter your grades : " );
    scanf ( "%d" , &grade);
    sum += grade;
    n++;
}
sum = sum - grade; // remove the last data .
n--; // Remove the last data .
// Calculate the average and print it to the screen .
average = sum / n;
printf ( " The average of the grades is %f .\n" , average);

return 0;
}
```

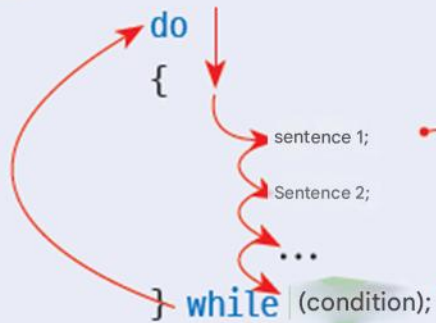
```
Enter
Enter your grade : 10
Enter your grade : 20
Enter your grade : 30
Enter your grade : 40
Enter your grade : 50
Enter your grade : -1
The average score is 30.000000 .
```

# do...while statement

## Syntax

do...while statement

yes

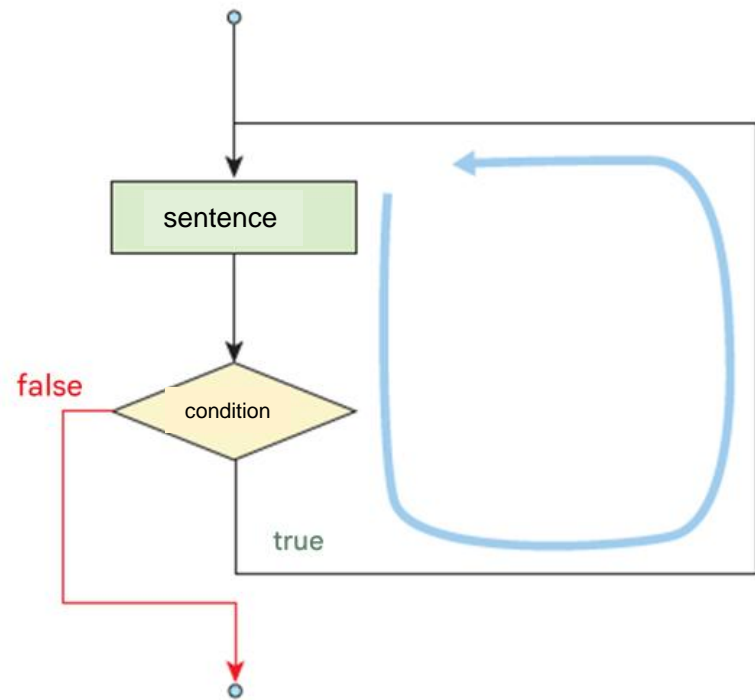
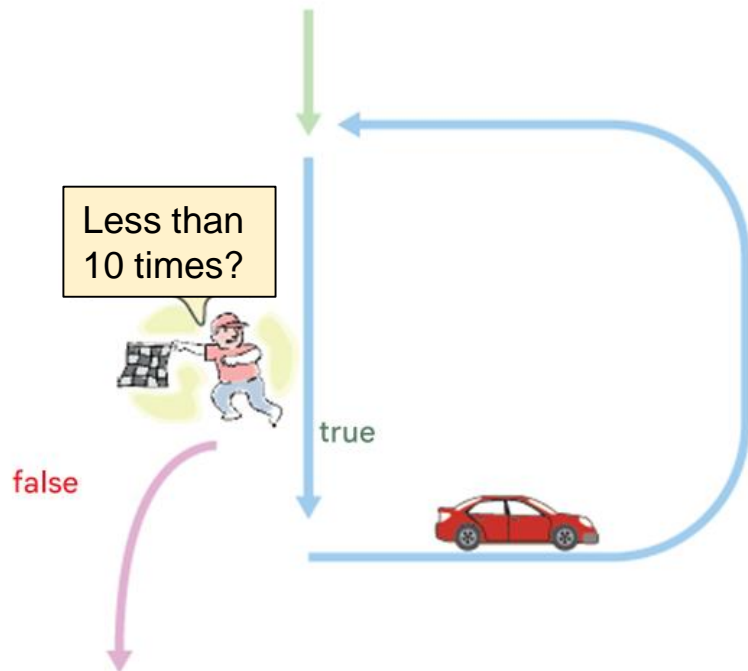


Repeated sentence

If the condition is true, the loop continues.


# do-while statement

- Executes the loop statement at least once .



# Example #1

- Let's write a program that adds the entered numbers until the user enters 0 using the do...while statement .



```
Enter an integer : 10  
Enter an integer : 20  
Enter an integer : 30  
Enter an integer : 0  
Sum of numbers = 60
```



# Example #1

```
// Add numbers until the user enters 0 .
#include <stdio.h>
int main( void )
{
    int number, sum = 0;


    // The loop body is executed at least once .
    do
    {
        printf ( " Enter an integer : " );
        scanf ( "%d" , &number);
        sum += number;
    } while (number != 0);

    printf ( " Sum of numbers = %d \n" , sum);

    return 0;
}
```

# Example #2

- do..while statement is widely used in input processing.



```
1--- Create new  
2--- Open file  
3--- Close file  
Select one : 1  
Selected menu =1
```

# Example #2

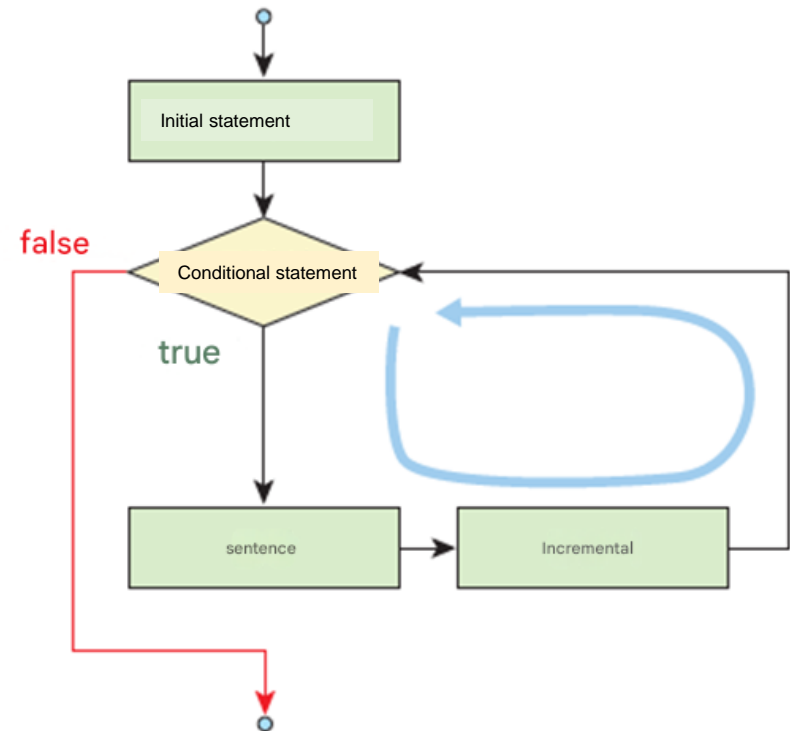
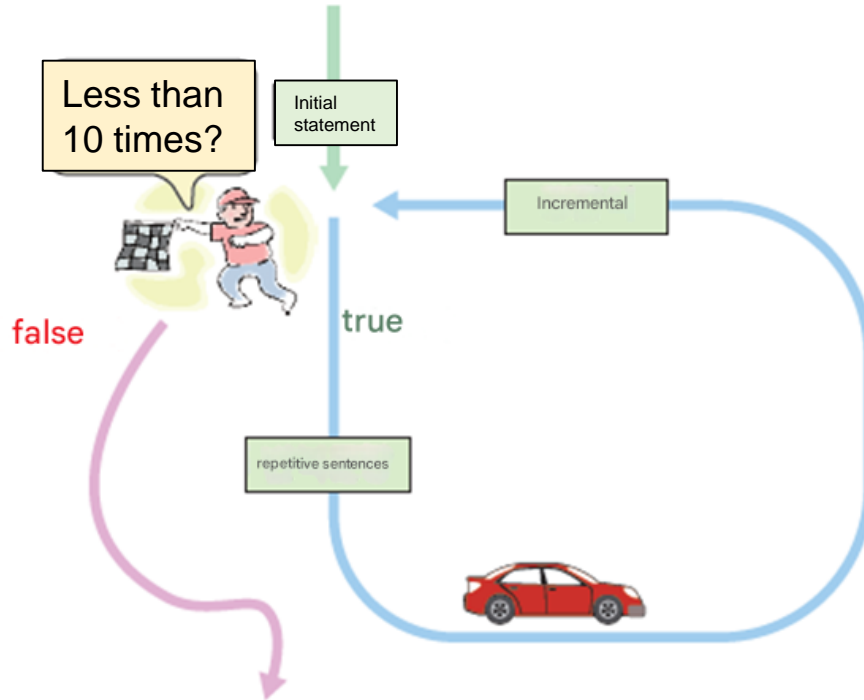
```
// Menu using do..while statement
#include <stdio.h>

int main( void )
{
    int i = 0;
    do
    {
        printf( "1--- New \n" );
        printf( "2--- Open file \n" );
        printf( "3--- Close file \n" );
        printf( " Choose one .\n" );
        scanf( "%d" , &i);
    } while (i < 1 || i > 3);

    printf( " Selected menu =%d\n" ,i);
    return 0;
}
```

# for loop

- A structure that repeats a set number of times



# Structure of for statement

Syntax

for 문

예

Init statement

Condition

Increase/decrease

```
for( i=0; i<5; i++ ) {  
    printf("Hello World!");  
} ;
```

반복되는 문장

# Initial expression, conditional expression, incremental expression

- Initial

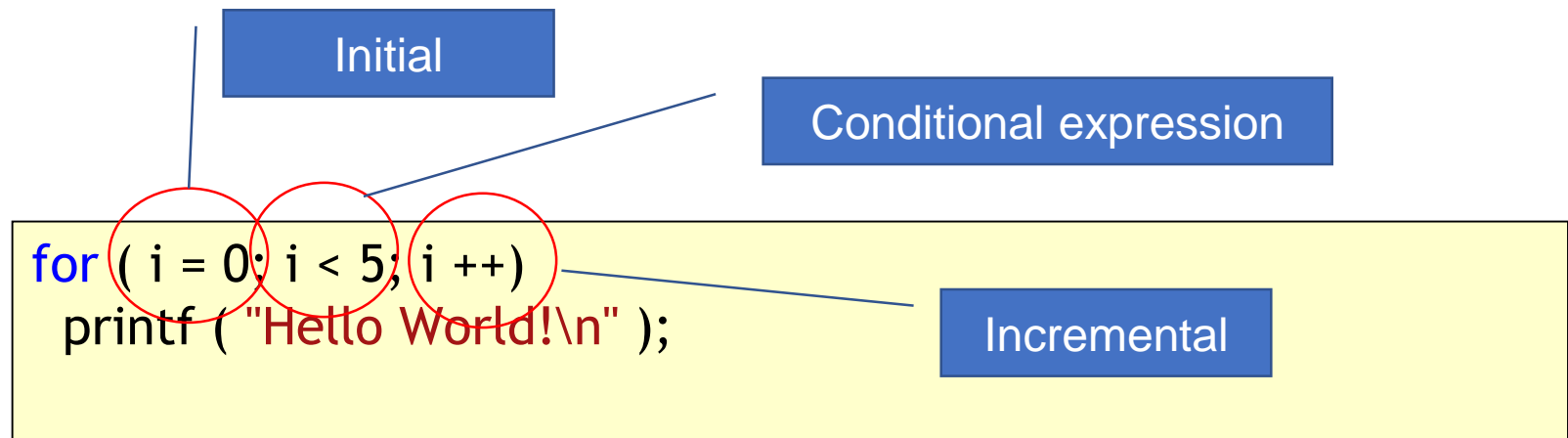
- The initialization expression is executed only once before starting the repeating loop. It is mainly used to initialize variable values .

- Conditional expression

- A formula that checks the condition for repetition. If the value of this expression becomes false, the repetition stops .

- Incremental

- After one loop execution is complete, the increment expression is executed.



# Example

```
Print
#include <stdio.h>

int main( void )
{
    int i;

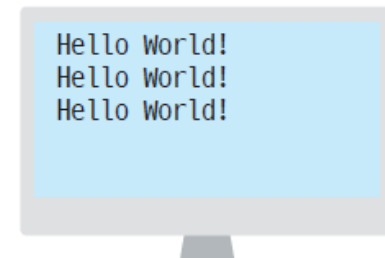
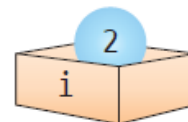
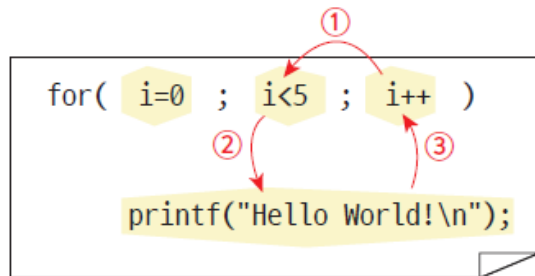
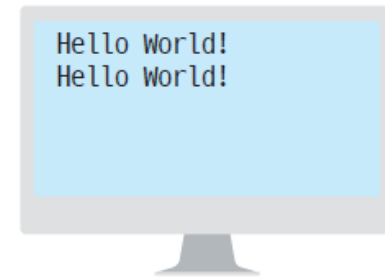
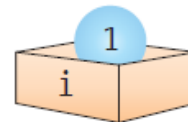
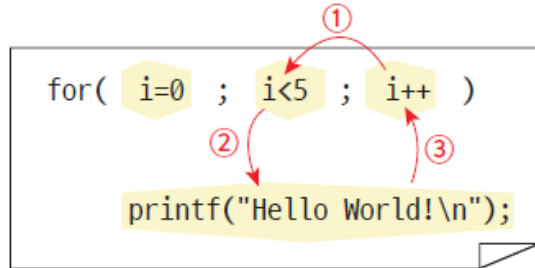
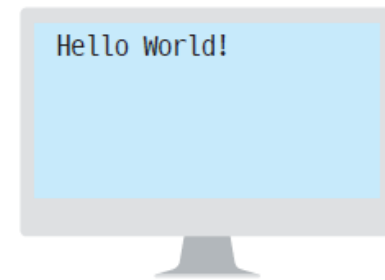
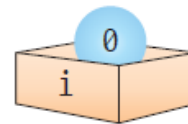
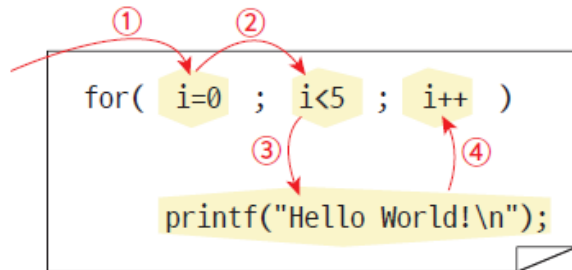
    for ( i = 0; i < 5; i ++ ) // i increases
        printf ( "Hello World!\n" );

    return 0;
}
```



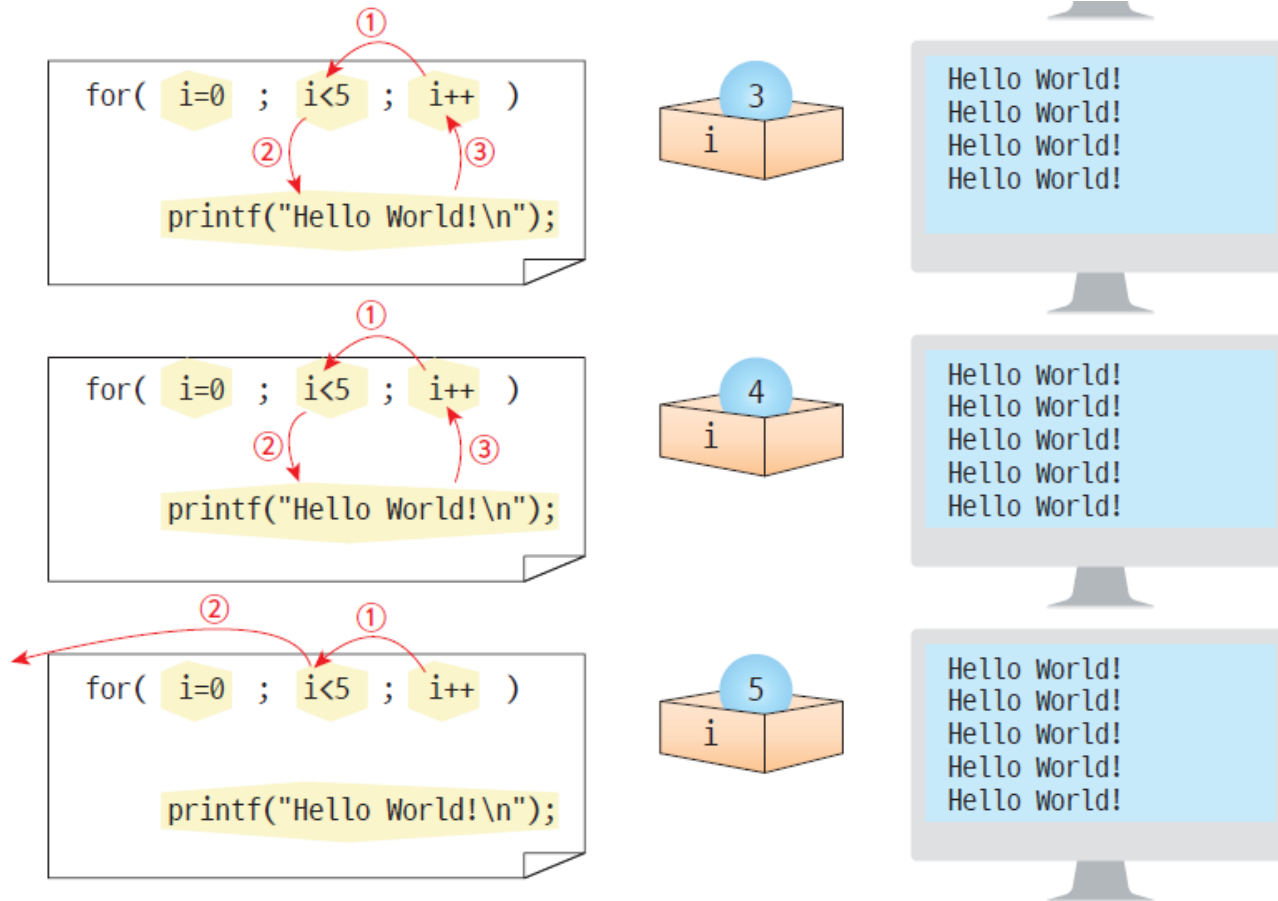
```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

# The execution process of "for"





# The execution process of "for"



## Example #2



Practice

- Let's write a program that adds integers from 1 to 10 and finds the sum .



Sum of integers from 1 to 10 = 55

# Example #2

```
// Integer sum program using repetition
#include <stdio.h>

int main( void )
{
    int i, sum;

    sum = 0;
    for (i = 1; i <= 10; i++)
        sum += i; // same as sum = sum + i;

    printf( " Sum of integers from 1 to 10 = %d\n" ,sum);

    return 0;
}
```

Sum of integers from 1 to 10 = 55

# Example #3

- Let's draw a box like the following on the screen using the \* character .



# Example #3

```
// Drawing a square using repetition
#include <stdio.h>

int main( void )
{
    int i;
    printf ( "*****" );

    for (i = 0; i < 5; i++)
        printf ( "* " );

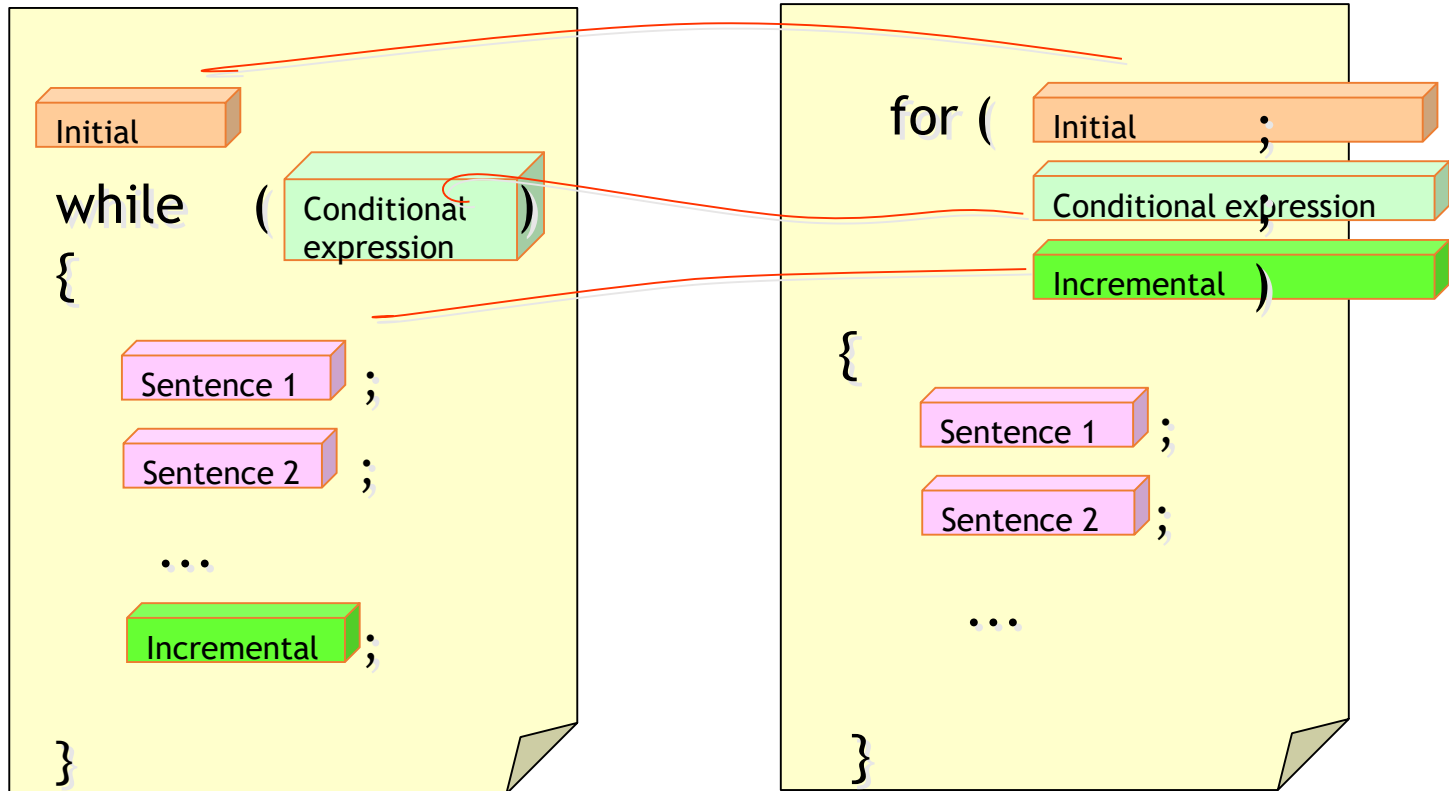
    printf ( "*****" );

    return 0;
}
```



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

# Relationship between while loop and for loop



Starting with C11, variables can be declared inside for loops.

```
for ( int i =0; i < 10; i ++ ) {  
    ...  
}
```

# Tip

Which of the three loops for, while, do...while should I use?

It's partly a matter of personal taste. A general criterion for choosing is that if you know how many times the loop will repeat, **a for loop is slightly more convenient than a while loop**. That is, you are less likely to forget to increment the loop control variable than with a while loop. **If there is only a condition and the exact number of repetitions is not known, a while structure is better**. If there are statements that must be **executed at least once, a do...while** structure is better.

Also, while and for are structures that check the condition before repeating, while do...while executes first and then checks the repeat condition. In general cases, not special cases, it is better to check the condition before repeating. It is the same as doing a thorough preliminary investigation before executing anything.





# Various forms of increasing and decreasing formulas

```
for ( int i = 10; i > 0; i -- )  
    printf ( "Hello World!\n" );
```

Using subtraction

```
for ( int i = 0; i < 10; i += 2 )  
    printf ( "Hello World!\n" );
```

Increase by 2

```
for ( int i = 1; i < 10; i *= 2 )  
    printf ( "Hello World!\n" );
```

Multiply by 2 .

```
for ( int i = 0; i < 100; i = ( i * i ) + 2 )  
    printf ( "Hello World!\n" );
```

Any formula is possible

# Various forms of increasing and decreasing formulas

```
for ( ; ; )  
    printf ( "Hello World!\n" );
```

Infinite loop

```
for ( ; i<100; i++ )  
    printf( "Hello World!\n" );
```

One part may be missing .

```
for ( i = 0, k = 0; i < 100; i++ )  
    printf( "Hello World!\n" );
```

two or more variables

```
for ( printf ( " loop start " ), i = 0; i < 100; i ++ )  
    printf ( "Hello World!\n" );
```

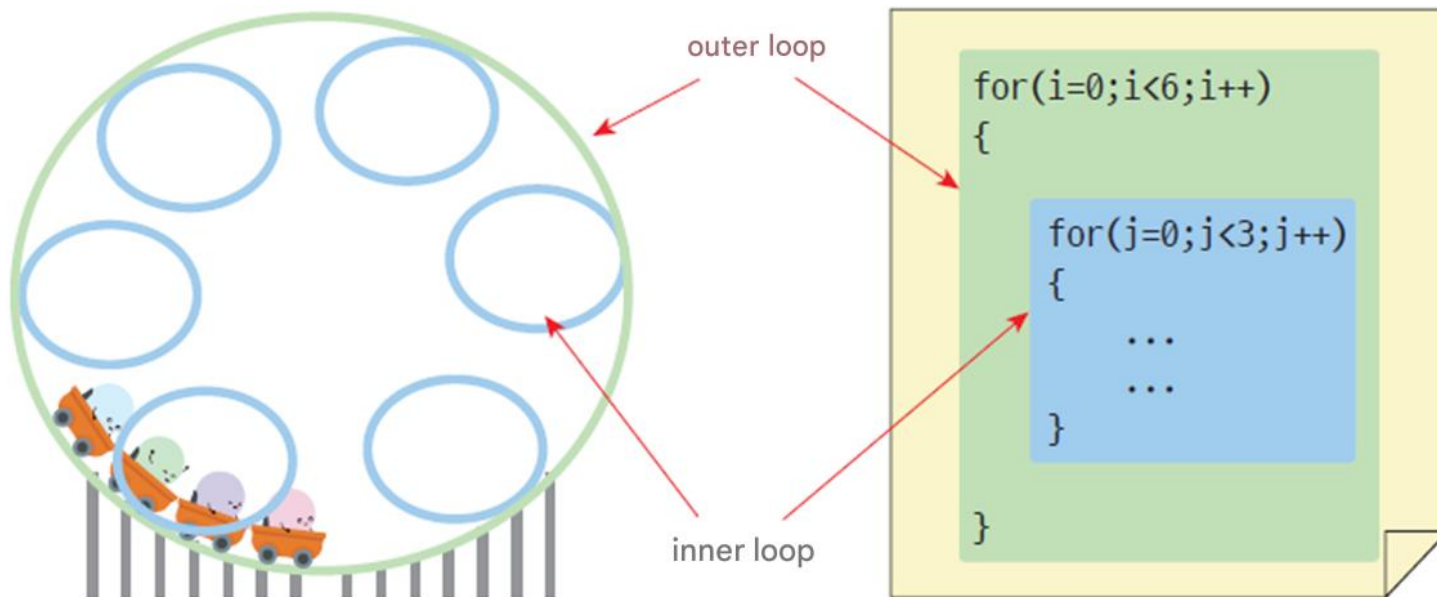
Any formula is possible

```
for ( i = 0; i < 100 && sum < 2000; i++ )  
    printf ( "Hello World!\n" );
```

Any complex expression can be a conditional expression.

# Nested loop

- loop : A loop that is located within another loop.



# Example #1



- The following example prints the \* symbol in a square shape



# Example #1

```
// Program to print * symbols in a square shape using
#include <stdio.h>

int main( void )
{
    int x, y;

    for (y = 0; y < 5; y++)
    {
        for (x = 0; x < 10; x++)
            printf ( "*" );
        printf ( "\n" );
    }

    return 0;
}
```



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

## Example #2

- Let's change the previous example a little so that it prints as follows . If you analyze the execution result in detail, you can see that y \* are printed on the yth line.



# Example #2

```
#include <stdio.h>
int main( void )
{
    int x, y;
    for (y = 1; y <= 5; y++)
    {
        for (x = 0; x < y; x++)
            printf ( "*" );
        printf ("\n" ); // Executes whenever
    }

    return 0;
}
```



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

# Infinite loop

- In a conditional control loop, sometimes the program repeats itself infinitely . This is known as an infinite loop . This is a problem because when an infinite loop occurs, the program cannot escape from it .
- But sometimes, intentionally Infinite loops are used because, for example, a traffic light control program must repeat infinitely.

## Syntax

infinite loop statement

### Syntax

```
while (1) {  
    if (condition)  
        break;    # Stop repetition.  
    if (condition)  
        continue; # Start the next iteration.  
}
```



# When infinite loops are useful

- It is especially used in cases where the conditions for exiting the loop are tricky. For example, let's say you need to exit the while loop if the number entered by the user is a multiple of 3 or a negative number .

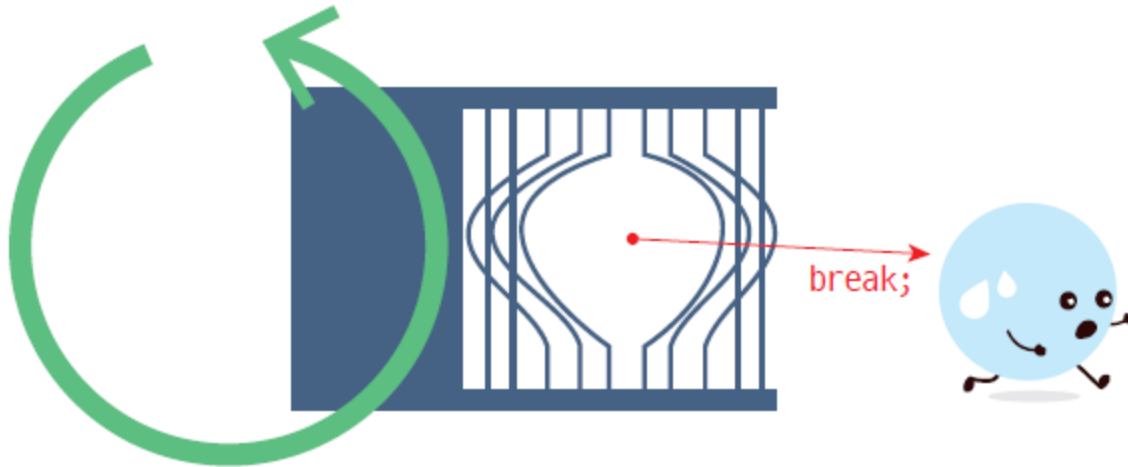
```
while((x % 3 != 0) && (x >= 0)) {  
    ...  
    ...  
    ...  
}
```



```
while (1) {  
    if (x%3 == 0) break;  
    if (x<0) break;  
    ...  
}
```

# break statement

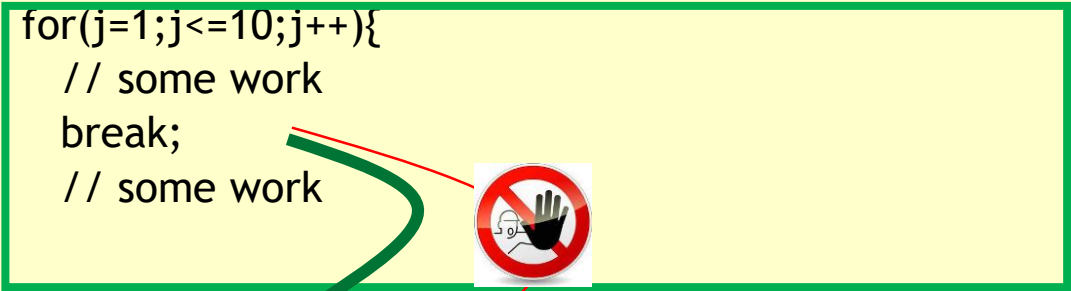
- break statement is used to exit a repeating loop .



# The only case you need "goto" statement

- If a problem occurs inside a nested loop , you can use goto to quickly exit the loop .
- break If you use it , you can only break one loop .

```
for( i =0;i<10;i++){  
    for(j=1;j<=10;j++){  
        // some work  
        break;  
        // some work  
    }  
}
```



# goto statement

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

```
int main( void )  
{
```

```
    int x, y;
```

```
    for (y = 1; y < 10000; y++)  
    {
```

```
        for (x = 1; x < 50; x++)  
        {
```

```
            if ( _kbhit () ) goto OUT;
```

```
            printf ( "*" );
```

```
        }
```

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

```
    }
```

```
OUT:
```

```
    return 0;
```

```
}
```

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

# continue statement

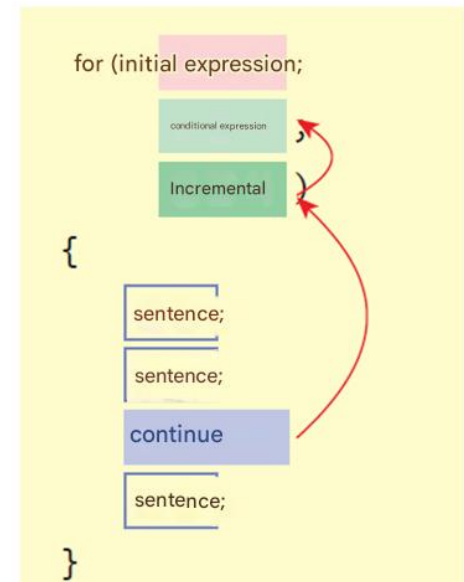
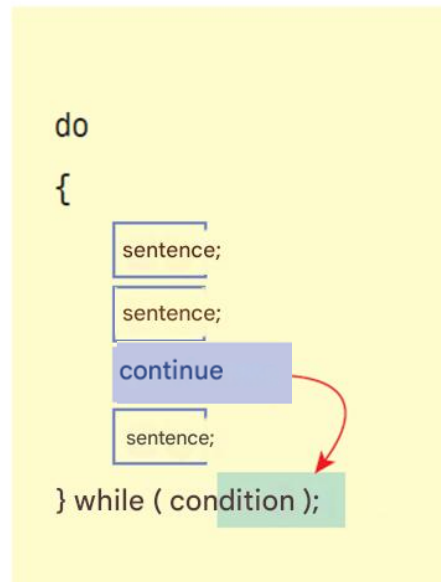
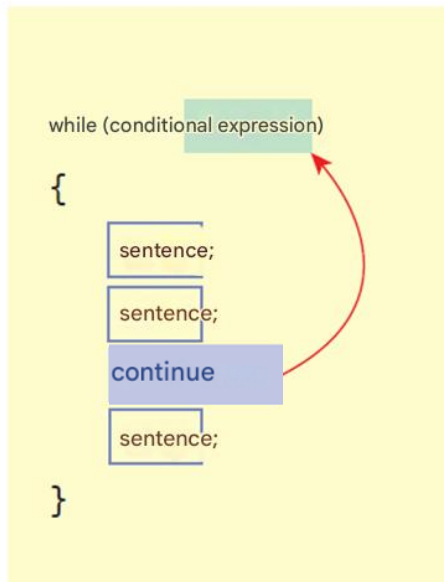
- Print all integers from 0 to 10, excluding those that are multiples of 3.

```
#include <stdio.h>

int main( void )
{
    int i;
    for ( i=0 ; i<10 ; i++ )
    {
        if ( i%3 == 0 )
            continue;
        printf ( "%d ", i );
    }
    return 0;
}
```

1 2 4 5 7 8

# continue statement



# Q & A

