SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.
An Autonomous Institution

COURSE NAME : 19ITT101 PROGRAMMING IN C & DATA STRUCTURES

I YEAR/ II SEMESTER

UNIT-II C DECISION STATEMENTS & FUNCTIONS

Topic: Looping Statements

Mrs.Vijayalakshmi.N
Assistant Professor
Department of Computer Science and Engineering

C Looping Statements

In any programming language including C, loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

The looping statements are used to execute a single statement or block of statements repeatedly until the given condition is FALSE.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1) It provides code reusability.

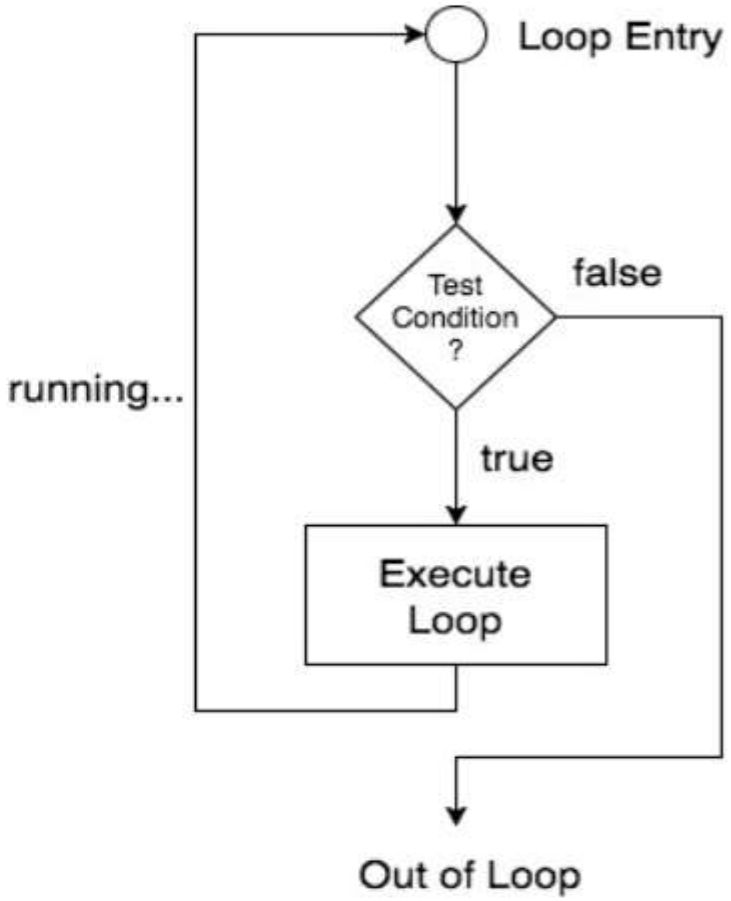
2) Using loops, we do not need to write the same code again and again.

3) Using loops, we can traverse over the elements of data structures (array or linked lists).

C Looping Statements

How it Works

The below diagram depicts a loop execution,



C Looping Statements

Types of Loops in C

Depending upon the position of a control statement in a program, looping in C is classified into two types:

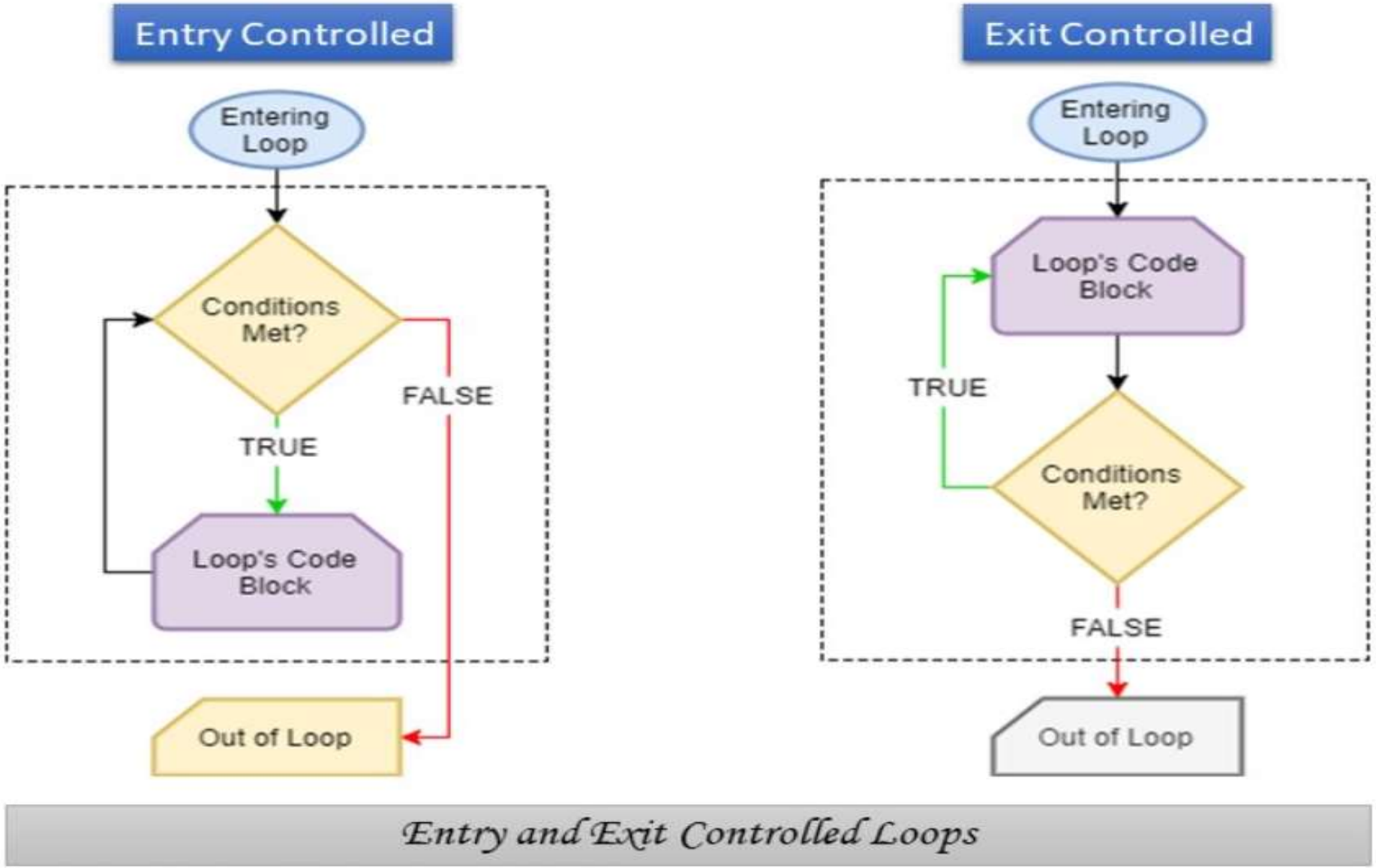
1. Entry controlled loop

2. Exit controlled loop

In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.

C Looping Statements



Entry and Exit Controlled Loops

C Looping Statements

Infinite Loop in C

An infinite loop is a looping construct that does not terminate the loop and executes the loop forever. It is also called an **indefinite** loop or an **endless** loop. It either produces a continuous output or no output.

Reason:

1. No termination condition is specified.

2. The specified conditions never meet.

The specified condition determines whether to execute the loop body or not.

C Looping Statements

Types of Loop

There are 3 types of Loop in C language, namely:

1. `while` loop
2. `for` loop
3. `do while` loop

while Loop

while Statement

The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE.

`while` loop can be addressed as an **entry control** loop. It is completed in 3 steps.

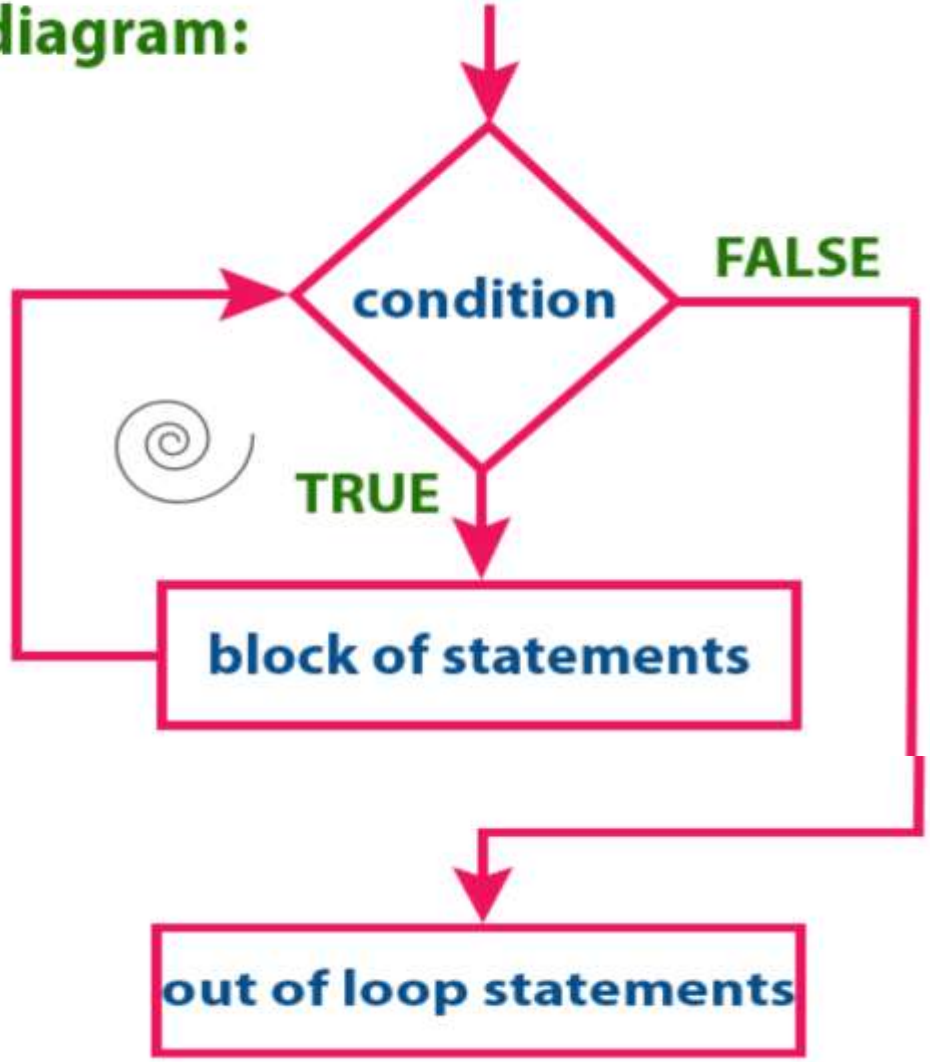
- Variable initialization.(e.g `int x = 0;`)
- condition(e.g `while(x <= 10)`)
- Variable increment or decrement (`x++` or `x--` or `x = x + 2`)

Syntax :

```
variable initialization;
while(condition)
{
    statements;
    variable increment or decrement;
}
```

while Loop

Execution flow diagram:



while Loop

Example 1: while loop

```c
// Print numbers from 1 to 5

#include <stdio.h>
int main()
{
    int i = 1;

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

    return 0;
}
```

while Loop

Output

```
1
2
3
4
5
```

Here, we have initialized `i` to 1.

1. When `i` is 1, the test expression `i <= 5` is true. Hence, the body of the `while` loop is executed. This prints 1 on the screen and the value of `i` is increased to 2.
2. Now, `i` is 2, the test expression `i <= 5` is again true. The body of the `while` loop is executed again. This prints 2 on the screen and the value of `i` is increased to 3.
3. This process goes on until `i` becomes 6. When `i` is 6, the test expression `i <= 5` will be false and the loop terminates.

for Loop

`for` loop

`for` loop is used to execute a set of statements repeatedly until a particular condition is satisfied. We can say it is an **open ended loop.**. General format is,

```
for(initialization; condition; increment/decrement)
{
    statement-block;
}
```

In `for` loop we have exactly two semicolons, one after initialization and second after the condition. In this loop we can have more than one initialization or increment/decrement, separated using comma operator. But it can have only one **condition**.

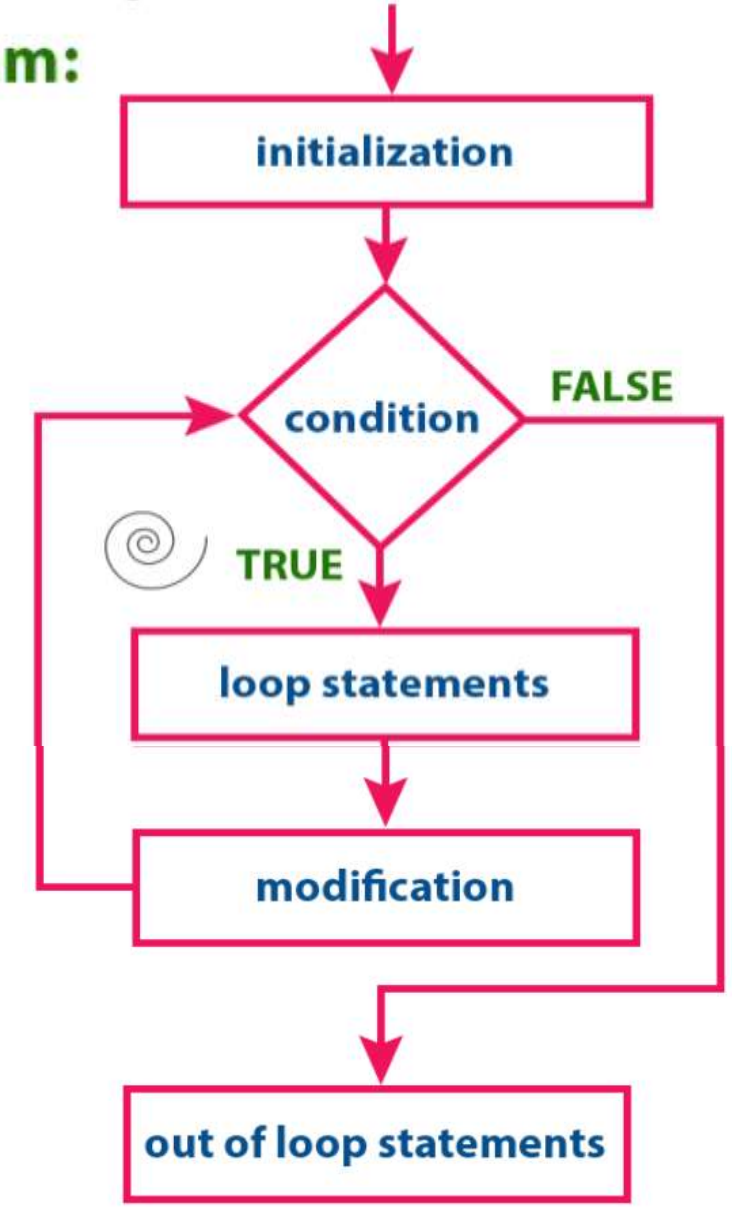
for Loop

The `for` loop is executed as follows:

1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is **true**, it executes the for-loop body.
4. Then it evaluate the increment/decrement condition and again follows from step 2.
5. When the condition expression becomes **false**, it exits the loop.

for Loop

Execution flow diagram:



for Loop

Example 1: for loop

```c
// Print numbers from 1 to 10
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

for Loop

Output

```
1 2 3 4 5 6 7 8 9 10
```

1. `i` is initialized to 1.
2. The test expression `i < 11` is evaluated. Since 1 less than 11 is true, the body of `for` loop is executed. This will print the **1** (value of `i`) on the screen.
3. The update statement `++i` is executed. Now, the value of `i` will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print **2** (value of `i`) on the screen.
4. Again, the update statement `++i` is executed and the test expression `i < 11` is evaluated. This process goes on until `i` becomes 11.
5. When `i` becomes 11, `i < 11` will be false, and the `for` loop terminates.

for Loop

Nested `for` loop

We can also have nested `for` loops, i.e one `for` loop inside another `for` loop. Basic syntax is,

```
for(initialization; condition; increment/decrement)
{
    for(initialization; condition; increment/decrement)
    {
        statement ;
    }
}
```

for Loop

Example: Program to print half Pyramid of numbers

```
#include<stdio.h>

void main( )
{
    int i, j;
    /* first for loop */
    for(i = 1; i < 5; i++)
    {
        printf("\n");
        /* second for loop inside the first */
        for(j = i; j > 0; j--)
        {
            printf("%d", j);
        }
    }
}
```

for Loop

OUTPUT:

```
1

21

321

4321

54321
```

do….while Loop

`do while` loop

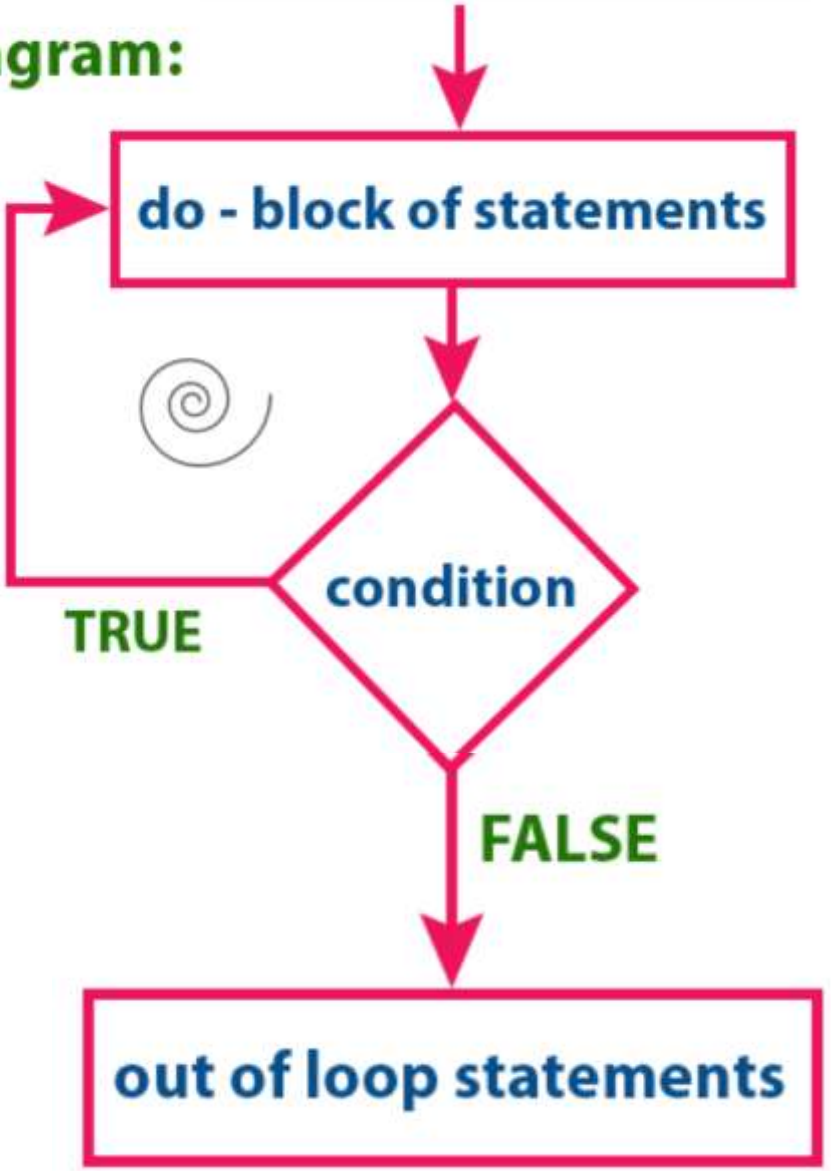
In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of `do-while` loop. `do` statement evaluates the body of the loop first and at the end, the condition is checked using `while` statement. It means that the body of the loop will be executed at least once, even though the starting condition inside `while` is initialized to be **false**. General syntax is,

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

The do-while statement is also known as the **Exit control looping statement**.

do….while Loop

Execution flow diagram:



do….while Loop

Example: Program to print first 10 multiples of 5.

```c
#include<stdio.h>

void main()
{
    int a, i;
    a = 5;
    i = 1;
    do
    {
        printf("%d\t", a*i);
        i++;
    }
    while(i <= 10);
}
```

OUTPUT:

```
5 10 15 20 25 30 35 40 45 50
```

Looping

Jumping Out of Loops

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes **true**. This is known as jumping out of loop.

C break

The break statement ends the loop immediately when it is encountered. Its syntax is:

```
break;
```

The break statement is almost always used with `if...else` statement inside the loop.

Looping

How break statement works?

```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

Looping

Example 1: break statement

```c
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, the loop terminates

#include <stdio.h>

int main() {
   int i;
   double number, sum = 0.0;

   for (i = 1; i <= 10; ++i) {
      printf("Enter a n%d: ", i);
      scanf("%lf", &number);

      // if the user enters a negative number, break the loop
      if (number < 0.0) {
         break;
      }

      sum += number; // sum = sum + number;
   }

   printf("Sum = %.2lf", sum);

   return 0;
}
```

Looping

Output

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

This program calculates the sum of a maximum of 10 numbers. Why a maximum of 10 numbers? It's because if the user enters a negative number, the `break` statement is executed. This will end the `for` loop, and the `sum` is displayed.

In C, `break` is also used with the `switch` statement.

Looping

C continue

The `continue` statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

```
continue;
```

The `continue` statement is almost always used with the `if...else` statement.

Looping

How continue statement works?

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Looping

Example 2: continue statement

```c
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, it's not added to the result

#include <stdio.h>
int main() {
   int i;
   double number, sum = 0.0;

   for (i = 1; i <= 10; ++i) {
      printf("Enter a n%d: ", i);
      scanf("%lf", &number);

      if (number < 0.0) {
         continue;
      }

      sum += number; // sum = sum + number;
   }

   printf("Sum = %.2lf", sum);

   return 0;
}
```

Looping

Output

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

In this program, when the user enters a positive number, the sum is calculated using `sum += number;` statement.

When the user enters a negative number, the `continue` statement is executed and it skips the negative number from the calculation.


Thank You!