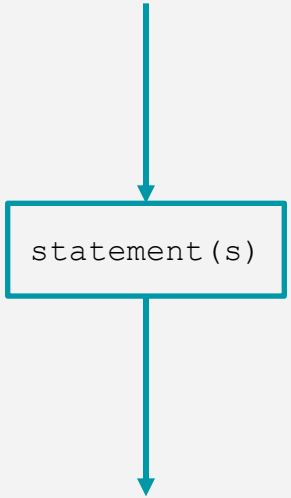
FLOW CONTROL IN C

Fabio Stroppa

Statements

Flow chart:



Statements

A **statement** is each of the individual instructions of a program. Statements always end with a semicolon **;** and are executed in the same order in which they appear in a program.

```
statement;
```

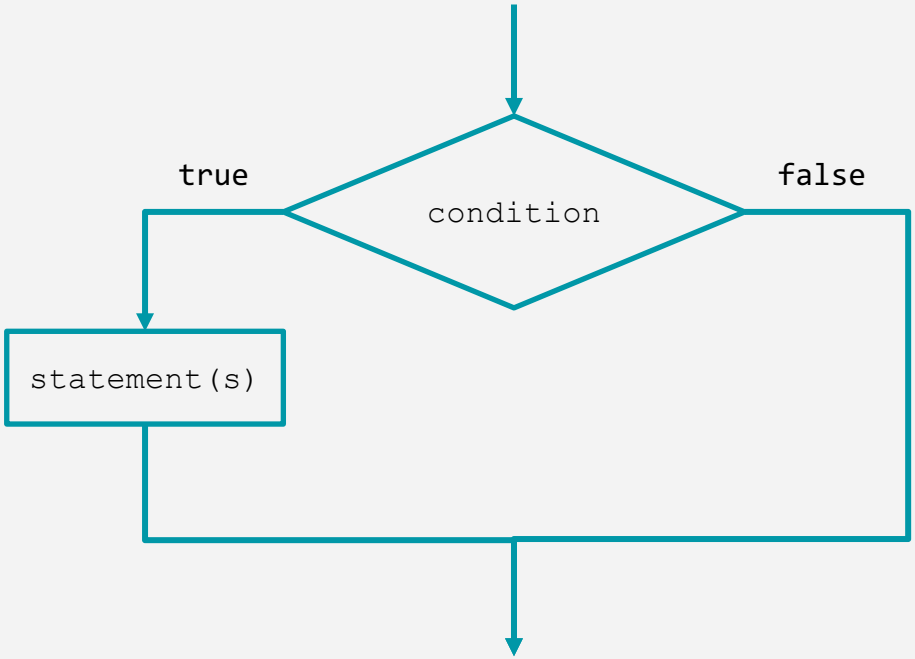
A **compound statement** is a group of statements (each of them terminated by its own semicolon), but all grouped together in a block, enclosed in curly braces: **{}**

```
{ statement1; statement2; statement3; }
```

compound statements

Selection statements: if

Flow chart:



Selection statements: if

The **`if`** keyword is used to execute a statement or block, if, and only if, a condition is fulfilled.

```
if (condition) statement
```

If `condition` is **`true`**, then execute `statement`.

```c
if (x == 100)
{
    printf("x is ");
    printf("100");
}
```

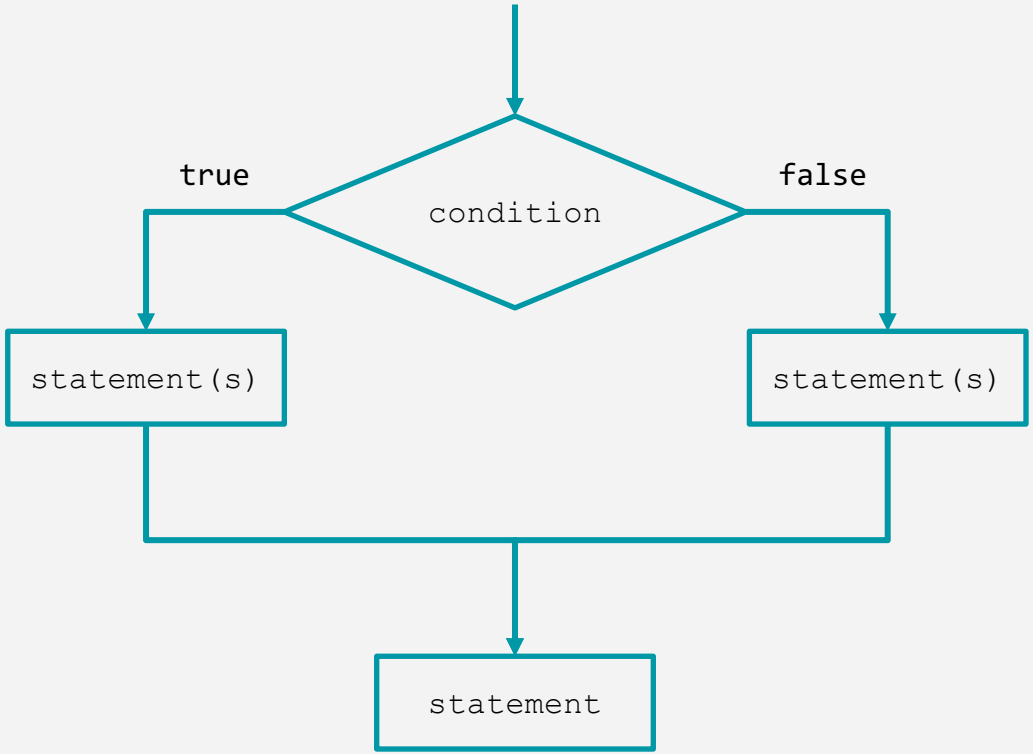
Use braces with compound statement within if statement

```c
if (x == 100)
    printf("x is 100");
```

Braces not necessary with single statement within if statement

Selection statements: if and else

Flow chart:



Selection statements: if and else

The **else** keyword is used to execute a statement or block, if, and only if, a condition is not fulfilled.

```
if (condition) statement1 else statement2
```

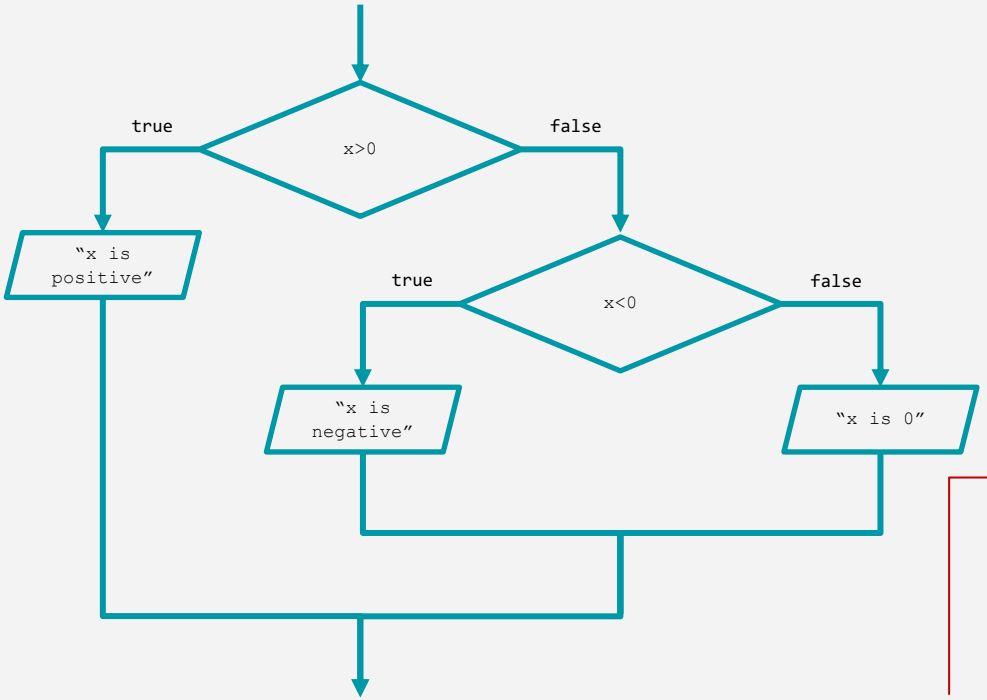
If `condition` is **`true`**, then execute `statement1`, otherwise execute `statement2`.

```
if (x == 100)
    printf("x is 100");
else
    printf("x is not 100");
```

```
if (x > 0)
    printf("x is positive");
else if (x < 0)
    printf("x is negative");
else
    printf("x is 0");
```

Several **if** + **else** structures can be concatenated with the intention of checking a range of values.

Selection statements: if and else




```
if (x > 0)
    printf("x is positive");
else if (x < 0)
    printf("x is negative");
else
    printf("x is 0");
```

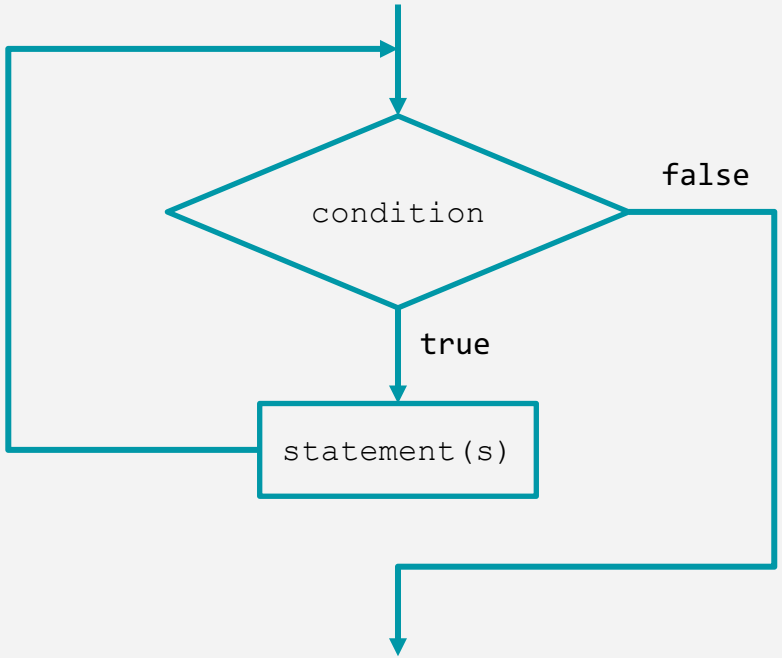
Several if + else structures can be concatenated with the intention of checking a range of values.

Iteration statements (loops)

Loops repeat a statement a certain number of times, or while a condition is fulfilled. They are introduced by the keywords **`while`**, **`do`**, and **`for`**.

Iteration statement: while loop

Flow chart:



Iteration statement: while loop

The while loop repeats `statement` while `condition` is **`true`**.

```
while (condition) statement
```

Note: with while loop, statements might never be executed. For example, if we had n = -1 (at line 7), the condition at the **`while`** loop (at line 8) would have been immediately false and the program would have skipped the loop (jumping directly to line 12).

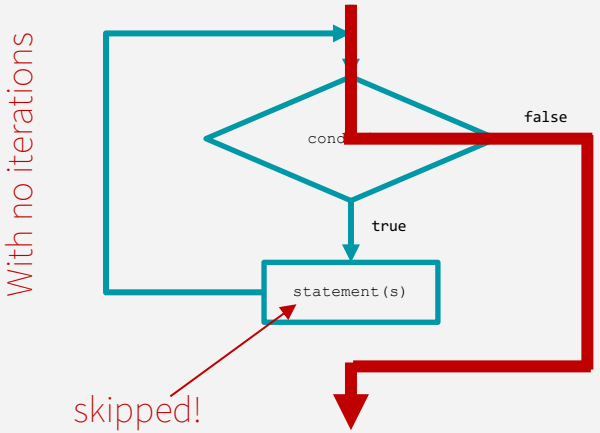
```
1:  // countdown using while loop
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:      int n = 10;
7:      while (n > 0) {
8:          printf("%d, ", n);
9:          --n;
10:     }
11:     printf("GO!\n");
12: }
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, GO!
```

Iteration statement: while loop

The while loop repeats `statement` while `condition` is **`true`**.

```
while (condition) statement
```



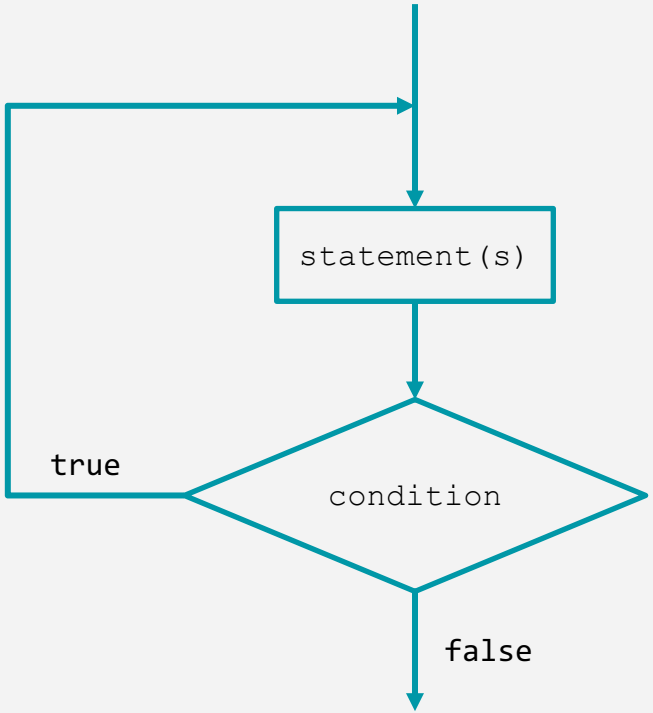

```c
// countdown using while loop
#include <stdio.h>

int main()
{
    int n = 10;
    while (n > 0) {
        printf("%d, ", n);
        --n;
    }
    printf("GO!\n");
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, GO!
```

Iteration statement: do-while loop

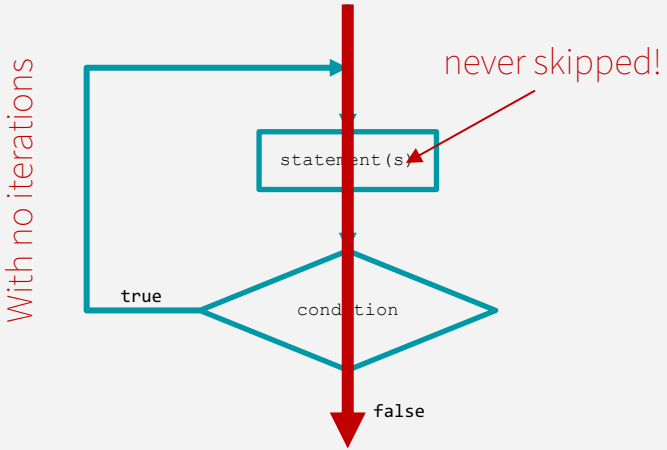
Flow chart:



Iteration statement: do-while loop

It behaves like a while loop, except that **`condition`** is evaluated after the execution of **`statement`** instead of before, guaranteeing at least one execution of **`statement`**, even if **`condition`** is never fulfilled.

```
do statement while (condition);
```




```c
//  echo machine
#include <stdio.h>

int main()
{
    int n;
    do {
        printf("Enter value: ");
        scanf("%d", &n);
        printf("%d\n", n);
    } while (n != 0);
}
```

Iteration statement: do-while loop

It behaves like a while loop, except that **`condition`** is evaluated after the execution of **`statement`** instead of before, guaranteeing at least one execution of **`statement`**, even if **`condition`** is never fulfilled.

```
do statement while (condition);
```

What does this code do?

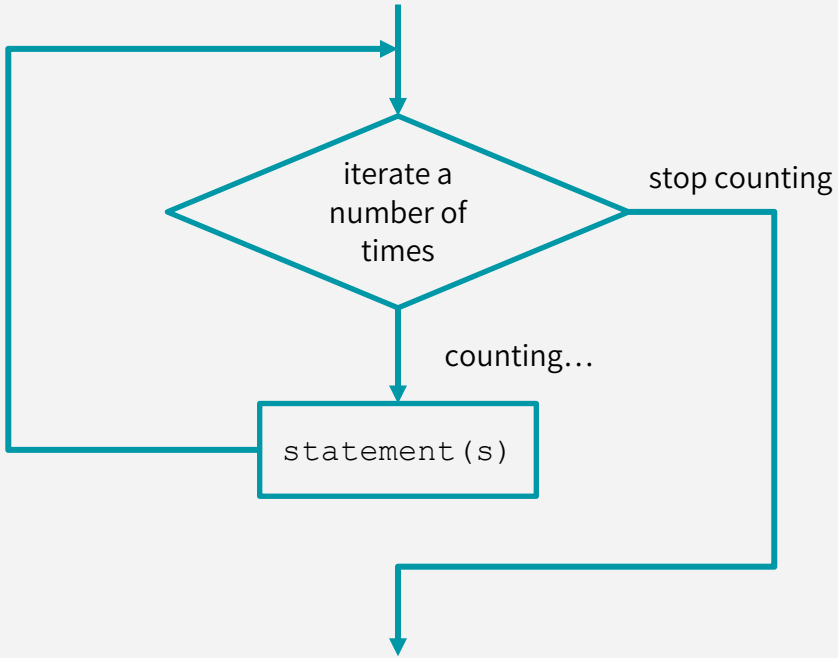
It allows users to enter numbers until they enter the value zero.

```c
//  echo machine
#include <stdio.h>

int main()
{
    int n;
    do {
        printf("Enter value: ");
        scanf("%d", &n);
        printf("%d\n", n);
    } while (n != 0);
}
```

Iteration statement: for loop

Flow chart:



Iteration statement: for loop

The for loop iterates **`statement`** a number of times. It provides specific locations to contain an **`initialization`** and an **`increase`** expression, executed before the loop begins the first time, and after each iteration, respectively.

```
for (initialization; condition; increase) statement;
```

```
1: //  countdown using a for loop
2: #include <stdio.h>
3:
4: int main()
5: {
6:    for (int n=10; n>0; n--)
7:       printf("%d, ", n);
8:    printf("GO!\n");
9: }
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, GO!
```

Iteration statement: for loop

The for loop iterates **`statement`** a number of times. It provides specific locations to contain an **`initialization`** and an **`increase`** expression, executed before the loop begins the first time, and after each iteration, respectively.

```
for (initialization; condition; increase) statement;
```

Note: the variable used to count the iteration is called *counter*. In this example, the variable **n** defined at line 7 in the **for** loop is a counter.

```
1: //  countdown using a for loop
2: #include <stdio.h>
3:
4: int main()
5: {
6:    for (int n=10; n>0; n--)
7:       printf("%d, ", n);
8:    printf("GO!\n");
9: }
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, GO!
```

Iteration statement: for loop

The for loop iterates **statement** a number of times. It provides specific locations to contain an **initialization** and an **increase** expression, executed before the loop begi ion, respectively.

ndition; increase) statement;

Not is
call n
defi

```
1:  // countdown using while loop
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int n = 10;
7:     while (n > 0) {
8:         printf("%d, ", n);
9:         --n;
10:    }
11:    printf("GO!\n");
12: }
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, GO!
```

```
1:  //  countdown using a for loop
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=10; n>0; n--)
7:        printf("%d, ", n);
8:     printf("GO!\n");
9:  }
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, GO!
```

Iteration statement: for loop

```
for (initialization; condition; increase) statement;
```

The three fields in a for-loop are optional. They can be left empty, but in all cases the semicolon signs **;** between them are required.

```
for ( ; n < 10 ; )
```

Equivalent to a **while** loop (there is only a condition)

```
for ( ; n < 10 ; ++n)
```

The counter **n** was probably initialized outside the for loop

```
for (int n = 0 ; ; ++n)
```

Infinite loop, equivalent to **while (true)**

Iteration statement: for loop

```
for (initialization; condition; increase) statement;
```

The three fields of a for loop can have more than one entry.

initializations condition increases

```
for ( int n = 0, i = 100; n != i; ++n, --i)
```

What does this loop do?

- **n** starts with a value of 0, and **i** with 100
- the condition is **n** not equal to **i**
- **n** is increased by 1, and **i** decreased by 1 on each iteration
- the loop's condition will become false after the 50th iteration, when both **n** and **i** are equal to 50

ANSWER: This loop will execute 50 times if neither **n** or **i** are modified within the loop.

Iteration statement: for loop

```
for (initialization; condition; increase) statement;
```

The three fields of a for loop can have more than one entry.

initializations | condition | increases

```
for ( int n = 0;  n < 50;  ++n )
```

What does this loop do?

- **n** starts with a value of 0, and **i** with 100
- the condition is **n** not equal to **i**
- **n** is increased by 1, and **i** decreased by 1 on each iteration
- the loop's condition will become false after the 50th iteration, when both **n** and **i** are equal to 50

ANSWER: This loop will execute 50 times if neither **n** or **i** are modified within the loop.

Jump statements

Jump statements allow the programmer to alter the flow of a program by performing jumps to specific locations.

They do not respect the rules of Böhm-Jacopini theorem; therefore, they produce unstructured code.

We will only use safe jump statements: **`break`** and **`continue`**.

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
1:  // break loop example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=5; n>0; n--) {
7:         printf("%d, ", n);
8:         if (n==3) {
9:            printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

`n = 5`

For loop starts.
Is **n** greater than **0**? **true**.

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
1:  // break loop example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=5; n>0; n--) {
7:        printf("%d, ", n);
8:        if (n==3) {
9:           printf("countdown aborted!");
10:          break;
11:       }
12:    }
13: }
```

`n = 5`

Print **n**.

```
5,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

`n = 5`

Is `n` equal to `3`? `false`.

```
5,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

`n = 5`

Is **n** equal to **3**? **false**.
Skip the statements within the **if** block.

```
5,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

`n = 4`

For loop decrement **n**.
Is **n** greater than **0**? **true**.

```
5,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
1:  // break loop example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=5; n>0; n--) {
7:        printf("%d, ", n);
8:        if (n==3) {
9:           printf("countdown aborted!");
10:          break;
11:       }
12:    }
13: }
```

`n = 4`

Print **n**.

```
5, 4,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

`n = 4`

Is `n` equal to `3`? `false`.

```
5,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

`n = 4`

Is **n** equal to **3**? **false**.
Skip the statements within the **if** block.

```
5,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

`n = 3`

For loop decrement **n**.
Is **n** greater than **0**? **true**.

```
5,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
1:  // break loop example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=5; n>0; n--) {
7:        printf("%d, ", n);
8:        if (n==3) {
9:           printf("countdown aborted!");
10:          break;
11:       }
12:    }
13: }
```

n = 3

Print **n**.

```
5, 4, 3,
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

```
5, 4, 3,
```

`n = 3`

Is `n` equal to `3`? `true`.

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
1:  // break loop example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=5; n>0; n--) {
7:        printf("%d, ", n);
8:        if (n==3) {
9:           printf("countdown aborted!");
10:          break;
11:       }
12:    }
13: }
```

```
5, 4, 3, countdown aborted!
```

n = 3

Print abort.

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
 1: // break loop example
 2: #include <stdio.h>
 3:
 4: int main()
 5: {
 6:    for (int n=5; n>0; n--) {
 7:        printf("%d, ", n);
 8:        if (n==3) {
 9:           printf("countdown aborted!");
10:           break;
11:        }
12:    }
13: }
```

```
5, 4, 3, countdown aborted!
```

`n = 3`

break stops the loop.

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```c
// break loop example
#include <stdio.h>

int main()
{
   for (int n=5; n>0; n--) {
       printf("%d, ", n);
       if (n==3) {
          printf("countdown aborted!");
          break;
       }
   }
}
```

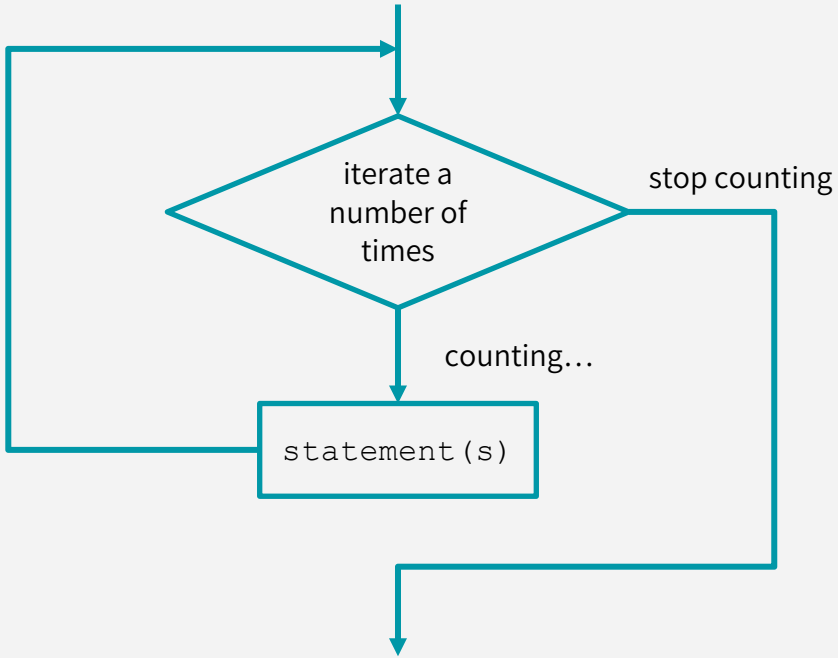
```
5, 4, 3, countdown aborted!
```

n = 3

End of program.

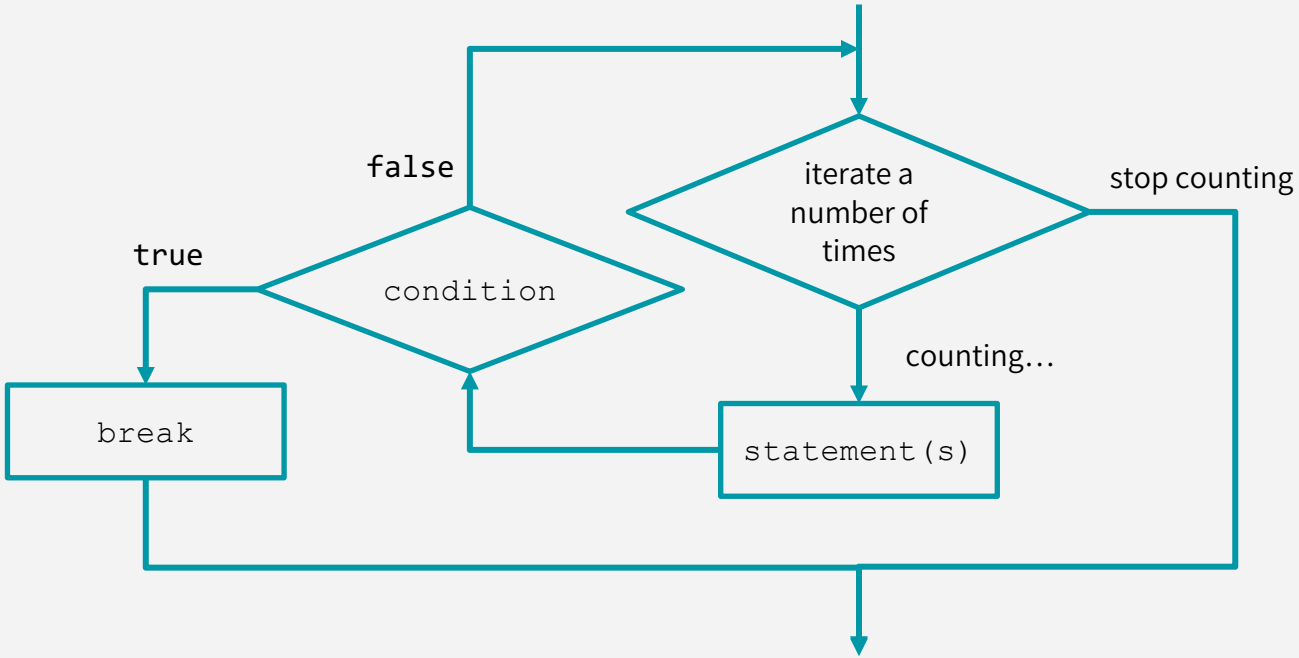
Jump statements: break is unstructured!

Flow chart:



Jump statements: break is unstructured!

Flow chart:



Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```c
// break loop example
#include <stdio.h>

int main()
{
   for (int n=5; n>0; n--) {
       printf("%d, ", n);
       if (n==3) {
          printf("countdown aborted!");
          break;
       }
   }
}
```

```
5, 4, 3, countdown aborted!
```

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
1:  // break loop example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=5; n>0; n--) {
7:        if (n>=3)
8:           printf("%d, ", n);
9:        if (n==3)
10:          printf("countdown aborted!");
11:    }
12: }
```

```
5, 4, 3, countdown aborted!
```

What is the difference between this code (no **break** statement) and the previous one (with **break** statement)?

Jump statements: break

It leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

```
1:  // break loop example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     for (int n=5; n>0; n--) {
7:        if (n>=3)
8:           printf("%d, ", n);
9:        if (n==3)
10:          printf("countdown aborted!");
11:    }
12: }
```

```
5, 4, 3, countdown aborted!
```

The difference is that the **break** will stop the **for** loop, while this code will still iterate until **n==0**.

So the countdown wasn't properly aborted… it just didn't produce any output.

Jump statements: continue

It causes the program to skip the rest of the loop in the current iteration, as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

```
1:  // break loop example
2:  #include <stdio.h>
4:
5:  int main()
6:  {
7:     for (int n=10; n>0; n--) {
8:        if (n==5) continue;
9:           printf("%d, ", n);
10:    }
11:    printf("GO!\n");
12: }
```

```
10, 9, 8, 7, 6, 4, 3, 2, 1, GO!
```

Jump statements: continue

It causes the program to skip the rest of the loop in the current iteration, as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

```
1:  // break loop example
2:  #include <stdio.h>
4:
5:  int main()
6:  {
7:     for (int n=10; n>0; n--) {
8:        if (n==5) continue;
9:           printf("%d, ", n);
10:    }
11:    printf("GO!\n");
12: }
```

5 was not printed!

```
10, 9, 8, 7, 6, 4, 3, 2, 1, GO!
```

Jump statements: continue

It causes the program to skip the rest of the loop in the current iteration, as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

```
1:  // break loop example
2:  #include <stdio.h>
4:
5:  int main()
6:  {
7:     for (int n=10; n>0; n--) {
8:        if (n!=5)
9:           printf("%d, ", n);
10:    }
11:    printf("GO!\n");
12: }
```

5 was not printed!

```
10, 9, 8, 7, 6, 4, 3, 2, 1, GO!
```

Jump statements: goto

JUST
DON’T

Selection statements: switch

Its purpose is to check for a value among a number of possible constant expressions. It is something similar to concatenating if-else statements but limited to constant expressions.

```
switch (expression) {
    case constant-value1:
          group-of-statements1;
          break;
    case constant-value2:
          group-of-statements2;
          break;
    .
    .
    .
    default:
          default-group-of-statements;
}
```

Switch evaluates `expression` and checks if it is equivalent to `constant-value1`; if it is, it executes `group-of-statements1` until it finds the break statement. When it finds this break statement, the program jumps to the end of the entire switch statement (the closing brace).

If `expression` was not equal to `constant-value1`, it is then checked against `constant-value2`. If it is equal to this, it executes `group-of-statements2` until a break is found, when it jumps to the end of the switch.

Finally, if the value of expression did not match any of the previously specified constants (there may be any number of these), the program executes the statements included after the `default:` label, if it exists (since it is optional).

Selection statements: switch

```c
// switch example
#include <stdio.h>

int main()
{
   int x;
   printf("Enter number: ");
   scanf("%d", &x);
   switch (x){
      case 1:
           printf("ONE\n");
           break;
      case 2:
           printf("TWO\n");
           break;
      default:
           printf("Neither ONE nor TWO\n");
   }
}
```

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
```

User enters number **1**, stored in **x**.

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
```

switch statement evaluates **x**.

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
```

Is **x** equal to 1?

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
```

Is **x** equal to 1? **True**

Enter the **case 1** block.

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
ONE
```

Is **x** equal to 1? **True**

Enter the **case 1** block.

Print ONE.

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
ONE
```

break interrupts the flow, skipping the rest of the cases.

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
ONE
```

End of program.

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
ONE
```

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
ONE
```

```
Enter number: 2
TWO
```

Selection statements: switch

```c
// switch example
#include <stdio.h>

int main()
{
   int x;
   printf("Enter number: ");
   scanf("%d", &x);
   switch (x){
      case 1:
           printf("ONE\n");
           break;
      case 2:
           printf("TWO\n");
           break;
      default:
           printf("Neither ONE nor TWO\n");
   }
}
```

```
Enter number: 1
ONE
```

```
Enter number: 2
TWO
```

```
Enter number: 3
Neither ONE nor TWO
```

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

```
Enter number: 1
ONE
```

```
Enter number: 2
TWO
```

```
Enter number: 3
Neither ONE nor TWO
```

```
Enter number: 4
Neither ONE nor TWO
```

Selection statements: switch

```
1:  // switch example
2:  #include <stdio.h>
3:
4:  int main()
5:  {
6:     int x;
7:     printf("Enter number: ");
8:     scanf("%d", &x);
9:     switch (x){
10:       case 1:
11:            printf("ONE\n");
12:            break;
13:       case 2:
14:            printf("TWO\n");
15:            break;
16:       default:
17:            printf("Neither ONE nor TWO\n");
18:    }
19: }
```

The last statement does not need a **break**.

```
Enter number: 1
ONE
```

```
Enter number: 2
TWO
```

```
Enter number: 3
Neither ONE nor TWO
```

```
Enter number: 4
Neither ONE nor TWO
```