5.5 Programming loops

It is very often needed, that we want to repeat some part of code several times. In that case we can use programming loops where we can specify which code should be repeated. In general there are two very often situation where we are using the programming loops:

1. We know `how many times` some code should repeat and
2. The code is `repeated while the condition` is met.

5.5.1 For-Next Loop

So called `For-Next` loop is used whenever the repetition of the code can be controlled by a `counter`. Counter is a number with some **starting value** and gets incremented by each repetition of the code. When `counter` reaches the given **ending value** repetition will stop. Typical examples where `For-Next` loop is used are:

- filling an array of data,
- summarising of all the costs in the bill
- robot should turn for **8 times** with 45 degree step to complete full rotation.

5.5.2 While Loop

`While` loop is used in situations where we can not predict the numbers of repetitions in advanced. In this case we must state the `condition` that must be met to repeat the code. The repetition of the code will be terminated when the `condition` will not hold anymore. Typical examples are:

- read the content to end of file,
- divide some number by 2 while we can,
- while no obstacle is in front of the robot it should drive forward

5.5.3 Do-While Loop

The Do-While loop in C++ programming is a control flow statement that executes a block of code at least once and then either repeatedly or until a particular condition is met. The condition is evaluated after the execution of the block of code. If the condition is true, the block of code is executed again. This repeats until the condition becomes false.

Here are three examples where a do-while loop can be suitable in programming a mobile robot:

- Navigating a Maze: A do-while loop can be used to control a robot to navigate through a maze by repeating movements (forward, turn left or right) until it finds the exit.
- Obstacle Avoidance: A do-while loop can be used to program a robot to continuously move in a particular direction until it detects an obstacle, then it changes direction.
- Searching for a Specific Object: A robot can be programmed using a do-while loop to keep searching in an environment until a particular object is found. This can be useful in search and rescue missions, or in a manufacturing setting where a robot is used to find and retrieve specific items.

5.5.4 Task: FOR-NEXT LOOP

1. For example the next prog. 1 repeats the functions **robotLeft()** and **robotRight()** for **10 times** and robot will do a funny "dancing" move.

Program 1: Programming Loops.

```
#include "RobotMovingFunctions.h"

void setup()
{
  setIOpins();
  // Repeating Left and Right movement
  // for 10 times to make a danging move
  for (int i = 0; i < 10; i++)
  {
    robotLeft();
    delay(100);
    robotRight();
    delay(100);
  }
  stopTheRobot();
}

void loop()
{

}
```

2. Experiment a bit more with such programming techniques and change some code:
   - value of `i`,
   - duration of `delay()` function,
   - add some other functions to the `for-next` loop…

5.5.5 Task: WHILE LOOP

3. Change the `for-next` loop with this `while` loop. Can you predict the result?

```
while ( 1 == 1 ){
    robotLeft();
    delay(100);
    robotRight();
    delay(100);
}
```

Presented `while` loop is not an useful example as the condition `( 1 == 1 )` will never change and will be always `true`. So, we created an infinite loop. `While` loop is far more usable if in the condition is some sensor's value, as we will see in next sections.

5.5.6 Questions:

1. Name the situation where `for-next` loop can be used.
2. What is the purpose of a `counter` in `for-next` loop?
3. What is the difference between `for-next` and `while` loops?

5.5.7 Summary:

Loops in C++ programming are used for flow control, allowing developers to execute a block of code repeatedly until a certain condition is met. There are three types of loops: - for, - while, and - do-while.

The `for` loop is typically used when the number of iterations is known. It contains an initializer, a condition, and an iterator. The `while` loop executes a block of code as long as the condition remains true. Unlike the `for` loop, the number of iterations in a `while` loop is indeterminate and depends on when the condition becomes false. The `do-while` loop is similar to the `while` loop but executes the block of code at least once before checking the condition. Loops are fundamental for flow control in C++, allowing for efficient and organized code execution.

5.5.7.1 For Loop:

Executes a block of code a specific number of times.

```
for (initialization; condition; increment) {
    // Code to execute
}
```

5.5.7.2 While Loop

Executes a block of code as long as a condition remains true.

```
while (condition) {
    // Code to execute
}
```

5.5.7.3 Do-While Loop

Similar to the while loop, but it executes the block of code at least once before checking the condition.

```
do {
    // Code to execute
} while (condition);
```

<++>

5.5.8 Issues:

5.5.8.1 *Can I measure the execution time of the loop?*

Yes, you can. You must save the time before the loop and save the time after the loop is executed. The difference in these two values is the spent in the execution of the loop. A minimal working example counld look like this:

```
unsigned long start_time = millis();
for (int i = 0; i<100; i++)
{
  //some code in this loop
}
unsigned long stop_time = millis();
unsigned long loop_duration = stop_time - start_time;
```

5.5.8.2 *Can I exit a while loop.*

Yes, you can use the “break” statement to exit a while loop in C++. However, this is not a common practice it is advised to set appropriate condition to exit a while loop. Here is an example of using “brake” statement:

```
int x = 0;
while (x < 10) {
  Serial.println(x);
  x++;
  if (x == 5) {
    break;
  }
}
```

This code will output the following to the serial port:

```
0
1
2
3
4
```

In this example, the “break” statement is used to exit the while loop when the value of “x” becomes 5. As a result, the loop only executes 5 times, rather than 10 times.

It is also possible to use the “continue” statement to skip the remainder of the current iteration of a loop, without exiting the loop entirely. For example:

```
int x = 0;
while (x < 10) {
  x++;
  if (x % 2 == 1) {
    continue;
  }
  Serial.println(x);
}
```

This code will output the following to the serial port:

```
2
4
6
8
10
```

In this example, the “continue” statement is used to skip the remainder of the current iteration of the loop if the value of “x” is odd. As a result, only the even values of “x” are printed.