Assignment I

Internet of Things

NAME : SARATH BHARATHI B

CLASS : I - MCA - A

TOPIC : Arduino Operators & Control Statements

DATE : 22/02/2023

Arduino Operators

The operators are widely used in Arduino programming from basics to advanced levels. It plays a crucial role in every programming concept like C, C++, Java, etc.

The operators are used to solve logical and mathematical problems. For example, to calculate the temperature given by the sensor based on some analog voltage.

The types of Operators classified in Arduino are:

- Arithmetic Operators
- Compound Operators
- Boolean Operators
- Comparison Operators
- Bitwise Operators

Arithmetic Operators

There are six basic operators responsible for performing mathematical operations in Arduino, which are listed below:

Assignment Operator (=) The Assignment operator in Arduino is used to set the variable's value. It is quite different from the equal symbol (=) normally used in mathematics.

Addition (+) The addition operator is used for the addition of two numbers. For example, P + Q.

Subtraction (-) Subtraction is used to subtract one value from the another. For example, P - Q.

Multiplication (*) The multiplication is used to multiply two numbers. For example, P * Q.

Division (/) The division is used to determine the result of one number divided with another. For example, P/Q.

Modulo (%) The Modulo operator is used to calculate the remainder after the division of one number by another number.

Most of the operators are similar to the usual operator used in mathematics.

Let's understand the operators with the help of two examples.

Example 1:

Consider the below code.

```
void setup ( )
{
Serial.begin( 9600 );
}
void loop ( )
{
b = 5 + 2;
Serial.println(b);
}
```

In the above code, we have assigned the result of the addition of two numbers to b before printing it to the console.

For output, click on the Upload and Serial Monitor button present on the toolbar.

```
 Output: 7
```

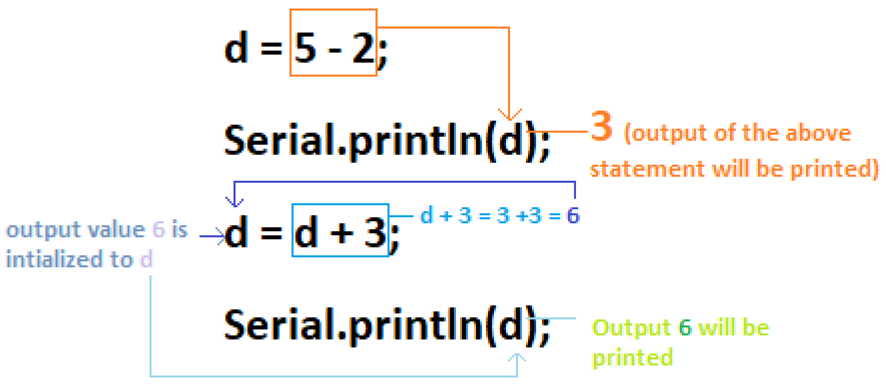
Example 2:

Consider the below code:

```
int d;
void setup ( )
{
Serial.begin( 9600 );
}
void loop ( )
{
d = 5 - 2;
Serial.println(d);
d = d + 3;
Serial.println(d);
}
```

Here, d= d +3 is not operated as a usual mathematical operation. It is the assignment operator where right of the function is evaluated first and is assigned to the left of the equal sign.

Let's consider the below image for better understanding.



Output:

```
3
6
```

Similarly, we can perform multiplication, modulo, and division. The int variable will store the integer values. For example, 20/3 = 6.

If we want decimal values to be printed, we need to use the float instead of int.

For example,

Consider the below code:

```
int b;
void setup ( )
{
Serial.begin( 9600 );
}
void loop ( )
{
b = 20.0 / 3; //  decimal value is used to force the compiler to print decimal
value.
Serial.println(b);
}
```

```
Output: 6.66
```

Order of mathematical operations

Let's understand the order of operations considered by the Arduino while performing calculation:

1. Parentheses ( )
2. Multiplication, division, and modulo

3. Addition and subtraction

If there are multiple operations next to each other, they will be computed from left to right.

Let's understand with an example.

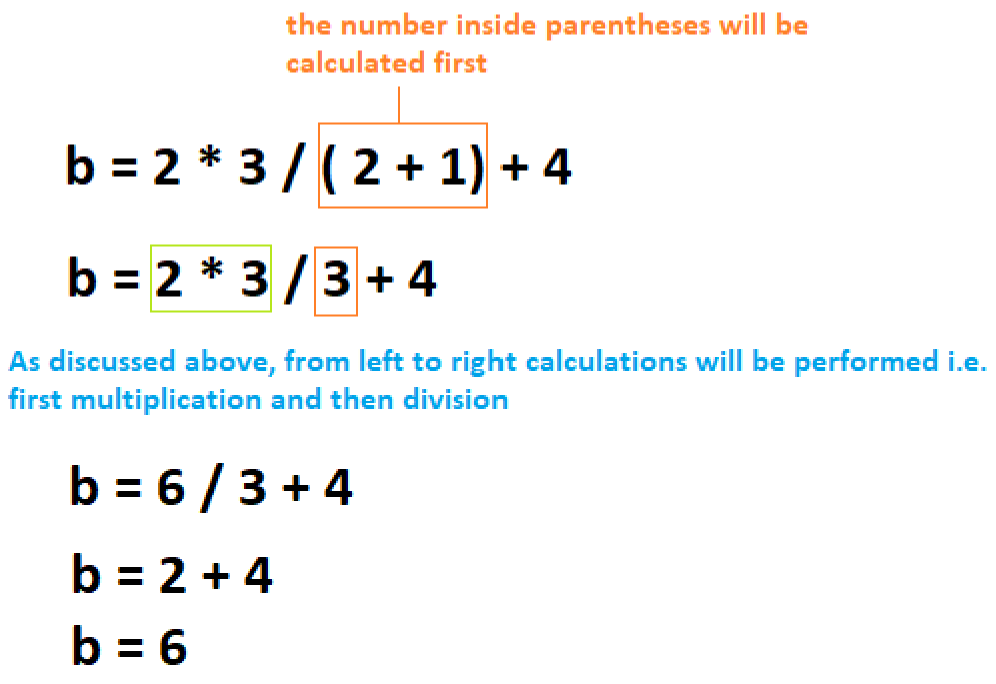
Consider the below code:

```
int c;
void setup ( )
{
Serial.begin( 9600 );
}
void loop ( )
{
c = 2 * 3 / (2 + 1)  + 4;
Serial.println(c);
}
```

Output:

```
6
```

Let's understand how the above output occurred. Consider the below image:


the number inside parentheses will be calculated first
b = 2 * 3 / (2 + 1) + 4
b = 2 * 3 / 3 + 4
As discussed above, from left to right calculations will be performed i.e. first multiplication and then division
b = 6 / 3 + 4
b = 2 + 4
b = 6

Compound Operators

The compound operators perform two or more calculations at once.

The result of the right operand is assigned to the left operand, as already discussed above. The same condition will apply to all the compound operators, which are listed below:

Let's consider a variable b.

b + +

- Here, b = b + 1. It is called the increment operator.

b + =

- For example, b + = 4. It means, b = b+ 4.

b - -

- Here, b = b - 1. It is called as the decrement operator.

b - =

- For example, b - = 3. It means, b = b - 3.

b * =

- For example, b * = 6. It means, b = b * 6.

b / =

- For example, b / = 5. It means, b = b / 5.

b % =

- For example, b % = 2. It means, b = b % 2.

Now, let's use the above operators with two variables, b and c.

- b + = c (b = b + c)
- b - = c (b = b - c)
- b * = c (b = b * c)
- b / = c (b = b / c)
- b % = c (b = b % c)

We can specify any variable instead of b and c.

Boolean Operators

The Boolean Operators are NOT (!), Logical AND (& &), and Logical OR (| |).

Let's discuss the above operators in detail.

Logical AND (& &)

The result of the condition is true if both the operands in the condition are true.

Consider the below example:

```
if ( a = = b & & b = = c )
```

Above statement is true if both conditions are true. If any of the conditions is false, the statement will be false.

Logical OR (| |)

The result of the condition is true, if either of the variables in the condition is true.

Consider the below example.

```
if ( a > 0 | | b  > 0 )
```

The above statement is true, if either of the above condition (a> 0 or b > 0) is true.

NOT (!)

It is used to reverse the logical state of the operand.

For example,

```
a ! = 2.
```

The NOT operator returns the value 1 or TRUE when the specified operand is FALSE. It also reverses the value of the specified expression.

Comparison Operators

The comparison operators are used to compare the value of one variable with the other.

The comparison operators are listed below:

less than (<) The less than operator checks that the value of the left operand is less than the right operand. The statement is true if the condition is satisfied.

Consider the below code.

```
int b;
int c ;
void setup ( )
{
Serial.begin( 9600 );
}
void loop ( )
{
b = 3;
c = 5;
if ( b < 4 )
Serial.println(b);
if ( c < 4)
Serial.println( c);
}
```

```
Output: 3
```

In the above code, if any of the two statement is correct, the corresponding value of the variable will be printed. Here, only first condition is correct. Hence, the value of b will be printed.

greater than (>)

The less than operator checks that the value of the left side of a statement is greater than the right side. The statement is true if the condition is satisfied.

For example, a > b.

If a is greater than b, the condition is true, else false.

equal to (= =)

It checks the value of two operands. If the values are equal, the condition is satisfied.

For example, a = = b.

The above statement is used to check if the value of a is equal to b or not.

not equal to (! =)

It checks the value of two specified variables. If the values are not equal, the condition will be correct and satisfied.

For example, a ! = b.

less than or equal to (< =)

The less or equal than operator checks that the value of left side of a statement is less or equal to the value on right side. The statement is true if either of the condition is satisfied.

For example, a < = b

It checks the value of a is less or equal than b.

greater than or equal to (> =)

The greater or equal than operator checks that the value of the left side of a statement is greater or equal to the value on the right side of that statement. The statement is true if the condition is satisfied.

For example, a > = b

It checks the value of a is greater or equal than b. If either of the condition satisfies, the statement is true.

Bitwise Operators

The Bitwise operators operate at the binary level. These operators are quite easy to use.

There are various bitwise operators. Some of the popular operators are listed below:

bitwise NOT (~)

The bitwise NOT operator acts as a complement for reversing the bits.

For example, if b = 1, the NOT operator will make the value of b = 0.

Let's understand with another example.

```
0 0 1 1 // Input or operand 1   ( decimal value 3)
1 1 0 0 // Output ( reverses the input bits ) decimal value is 12
```

bitwise XOR (^)

The output is 0 if both the inputs are same, and it is 1 if the two input bits are different.

For example,

```
1 0 0 1  // input 1 or operand 1
0 1 0 1  // input 2
1 1 0 0 // Output ( resultant - XOR)
```

bitwise OR (|)

The output is 0 if both of the inputs in the OR operation are 0. Otherwise, the output is 1. The two input patterns are of 4 bits.

For example,

```
1 1 0 0  // input 1 or operand 1
0 0 0 1  // input 2
1 1 0 1 // Output ( resultant - OR)
```

bitwise AND (&)

The output is 1 if both the inputs in the AND operation are 1. Otherwise, the output is 0. The two input patterns are of 4 bits.

For example,

```
1 1 0 0  // input 1 or operand 1
0 1 0 1  // input 2
0 1 0 0 // Output ( resultant - AND)
```

bitwise left shift (< <)

The left operator is shifted by the number of bits defined by the right operator.

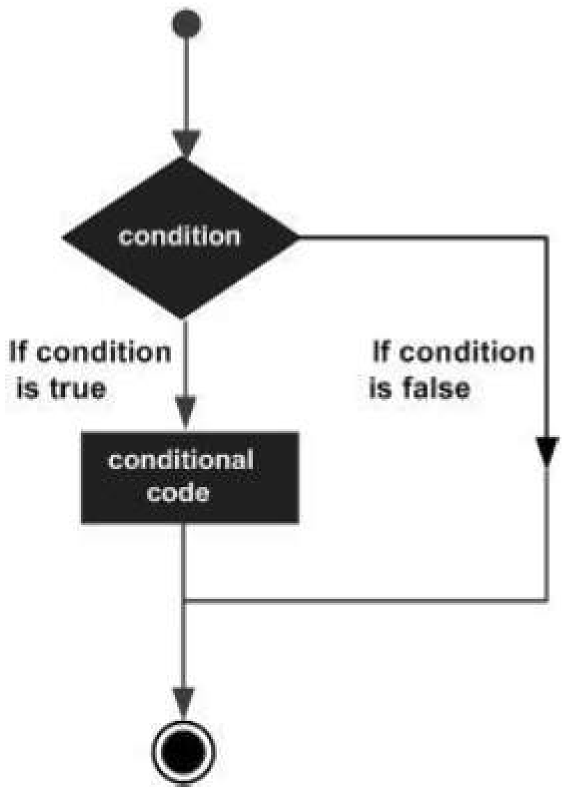
bitwise right shift (> >)

The right operator is shifted by the number of bits defined by the left operator.

Arduino - Control Statements

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program. It should be along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages −



Arduino - If statement

It takes an expression in parenthesis and a statement or block of statements. If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

Different forms of if statement Form 1

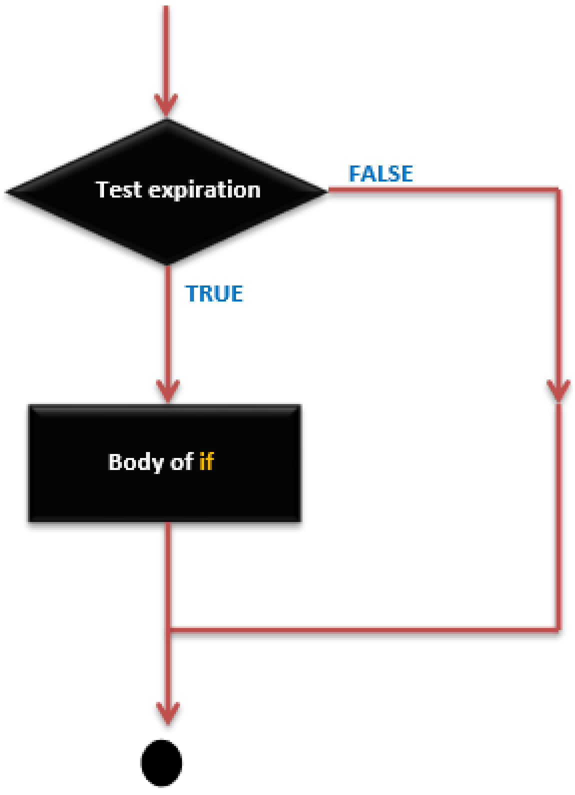
```
if (expression)
   statement;
```

You can use the if statement without braces { } if you have one statement.

Form 2

```
if (expression) {
   Block of statements;
}
```

if Statement – Execution Sequence



Example

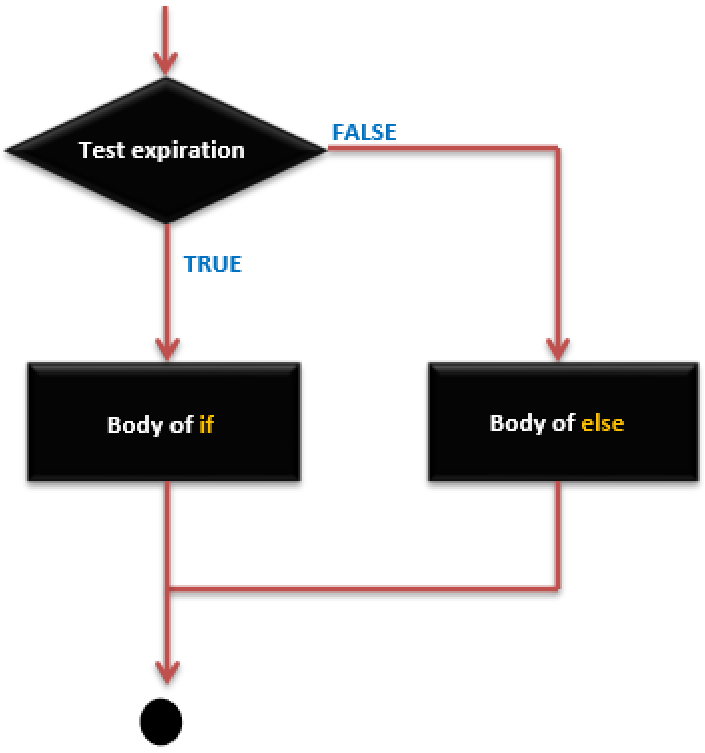
```
/* Global variable definition */
int A = 5 ;
int B = 9 ;

Void setup () {

}

Void loop () {
   /* check the boolean condition */
   if (A > B) /* if condition is true then execute the following statement*/
   A++;
   /* check the boolean condition */
   If ( ( A < B ) && ( B != 0 )) /* if condition is true then execute the
following statement*/ {
      A += B;
      B--;
   }
}
```

Arduino - If ...else statement

An if statement can be followed by an optional else statement, which executes when the expression is false.

if … else Statement Syntax

```
if (expression) {
   Block of statements;
}
else {
   Block of statements;
}
```



if…else Statement – Execution Sequence

Example

```
/* Global variable definition */
int A = 5 ;
int B = 9 ;

Void setup () {
}

Void loop () {
   /* check the boolean condition */
   if (A > B) /* if condition is true then execute the following statement*/ {
      A++;
   }else {
      B -= A;
   }
}
```

Arduino - If...else if ...else statement

The if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if...else if...else statements, keep in mind −

An if can have zero or one else statement and it must come after any else if's.

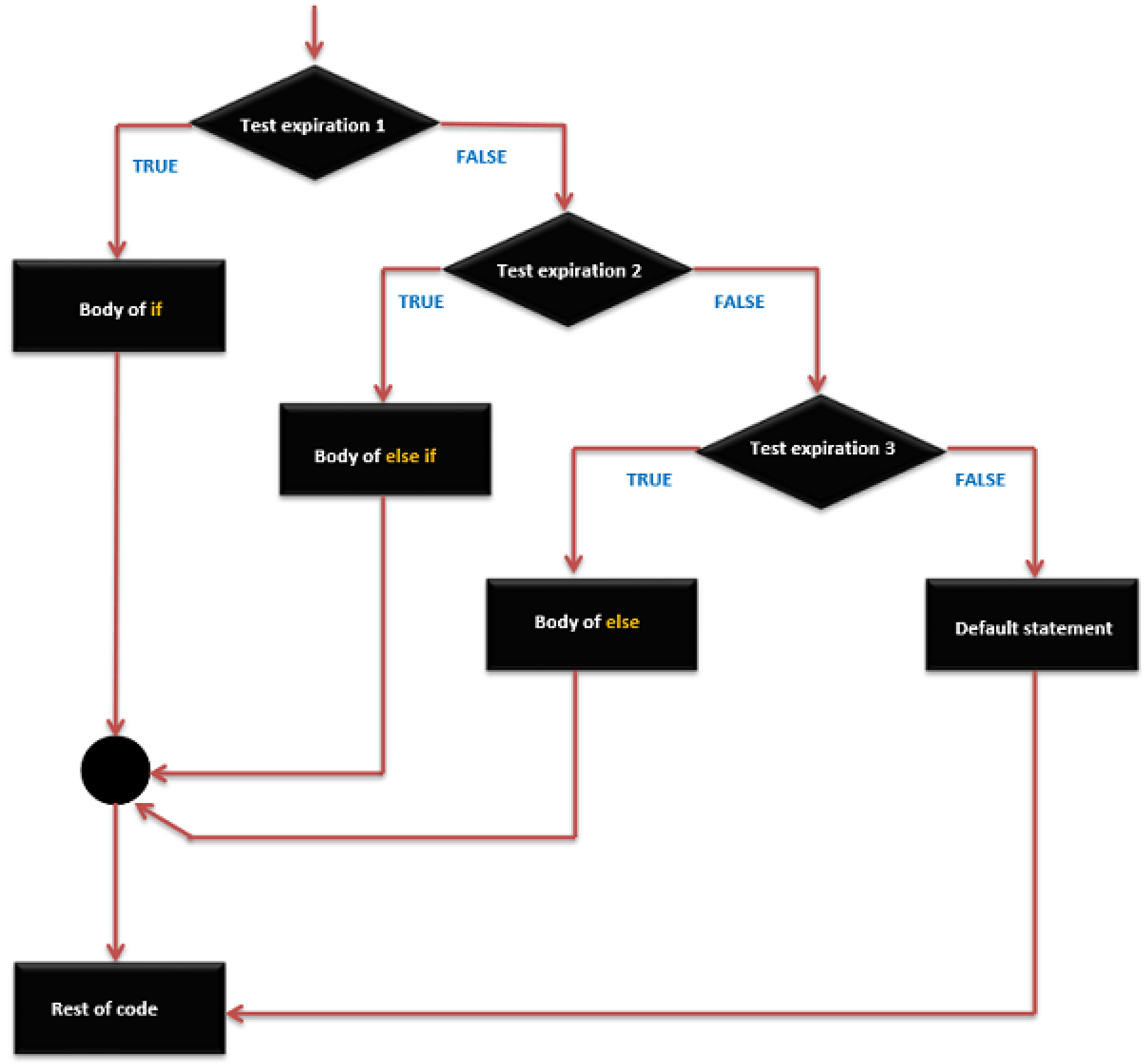
An if can have zero to many else if statements and they must come before the else.

Once an else if succeeds, none of the remaining else if or else statements will be tested.

if ... else if ...else Statements Syntax

```
if (expression_1) {
   Block of statements;
}

else if(expression_2) {
   Block of statements;
}
.
.
.

else {
   Block of statements;
}
```

if ... else if ... else Statement Execution Sequence



Example

```
/* Global variable definition */
int A = 5 ;
int B = 9 ;
int c = 15;

Void setup () {

}

Void loop () {
   /* check the boolean condition */
   if (A > B) /* if condition is true then execute the following statement*/ {
      A++;
   }
   /* check the boolean condition */
   else if ((A == B )||( B < c) ) /* if condition is true then
      execute the following statement*/ {
      C = B* A;
   }else
```

```
      c++;
  }
```

Arduino - switch case statement

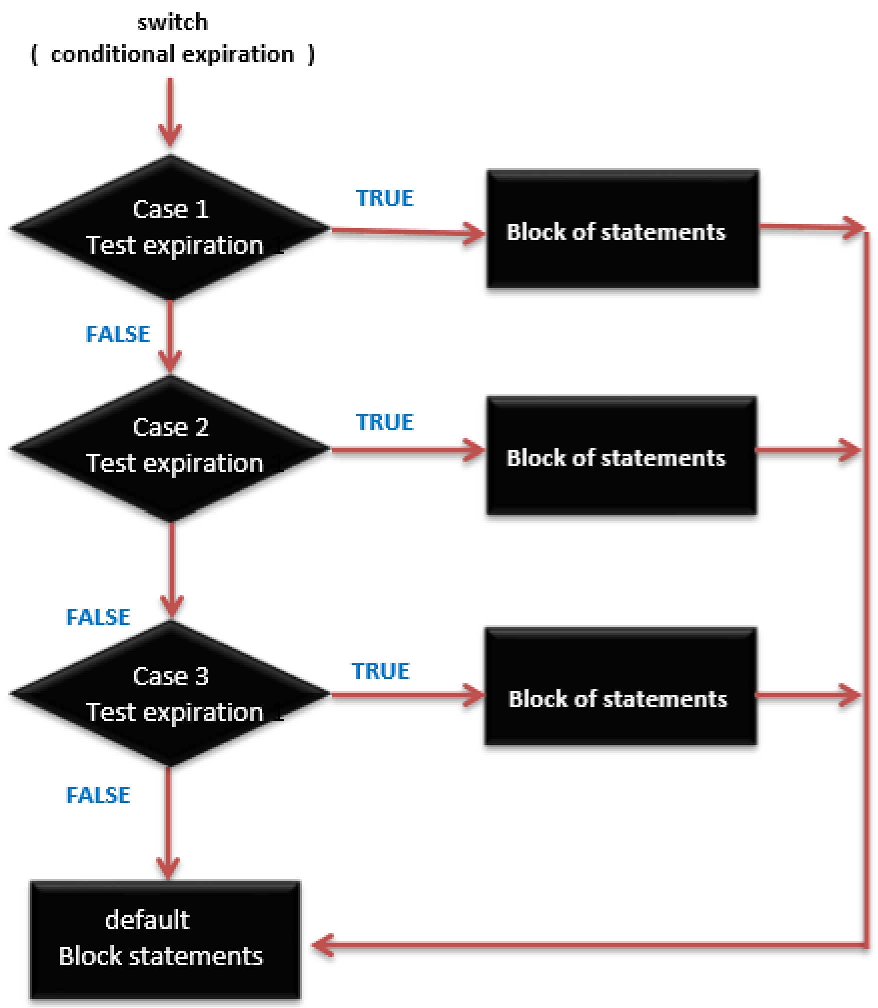
Similar to the if statements, switch...case controls the flow of programs by allowing the programmers to specify different codes that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in the case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The break keyword makes the switch statement exit, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

Switch Case Statement Syntax

```
switch (variable) {
   case label:
   // statements
   break;
}

case label: {
   // statements
   break;
}

default: {
   // statements
   break;
}
```

Switch Case Statement Execution Sequence



Example Here is a simple example with switch. Suppose we have a variable phase with only 3 different states (0, 1, or 2) and a corresponding function (event) for each of these states. This is how we could switch the code to the appropriate routine –

```
switch (phase) {
   case 0: Lo(); break;
   case 1: Mid(); break;
   case 2: Hi(); break;
   default: Message("Invalid state!");
}
```

Arduino - Conditional Operator ? :

The conditional operator ? : is the only ternary operator in C.

? : conditional operator Syntax

```
expression1 ? expression2 : expression3
```

Expression1 is evaluated first. If its value is true, then expression2 is evaluated and expression3 is ignored. If expression1 is evaluated as false, then expression3 evaluates and expression2 is ignored. The result will be a value of either expression2 or expression3 depending upon which of them evaluates as True.

Conditional operator associates from right to left.

Example

```
/* Find max(a, b): */
max = ( a > b ) ? a : b;
/* Convert small letter to capital: */
/* (no parentheses are actually necessary) */
c = ( c >= 'a' && c <= 'z' ) ? ( c - 32 ) : c;
```

Rules of Conditional Operator

- expression1 must be a scalar expression; expression2 and expression3 must obey one of the following rules.
- Both expressions have to be of arithmetic type.
- expression2 and expression3 are subjected to usual arithmetic conversions, which determines the resulting type.
- Both expressions have to be of void type. The resulting type is void.

Reference

- tutorialspoint.com/arduino_conditional_operator
- www.tutorialspoint.com/arduino_operators
- javatpoint.com/arduino-operators