"For" and "While"

Lesson 11

Welcome to the first Skill Level 3 lesson! You already know the basics from previous lessons. Now we're going to start learning some of the more powerful (and useful) things you can do with the C language.

Doing things over and over...

Sometimes you may want to do something several times. For example, let's say you want to blink Wink's eyes three times. You could write something like this...

```
eyesGreen(100);   //turn eyes on
delay(100);       //leave them on for 100ms
eyesOff();        //turn eyes off
delay(100);       //leave them off for 100ms

eyesGreen(100);   //turn eyes on
delay(100);       //leave them on for 100ms
eyesOff();        //turn eyes off
delay(100);       //leave them off for 100ms

eyesGreen(100);   //turn eyes on
delay(100);       //leave them on for 100ms
eyesOff();        //turn eyes off
delay(100);       //leave them off for 100ms
```

} Blink eyes

} Blink eyes again

} Blink eyes again

Sometimes it's quick to write a bit of code (like the first blink at the top), then copy and paste it two more times below. That's easy enough. But what if you want the eys to now blink eight times? You could quickly end up with a lot of copied and repeating code. On top of that, what if you want to change the speed of the blink? If you use the method above, you'll have a lot of delay statements to edit.

There is an easier way to do these things. It's called a "for" loop. It basically says "do the following thing 'for' this number of times". The for loop looks a bit strange at first but don't get too worried about that. It looks similar to the "if" control structure we learned in a previous lesson.

The example below does exactly the same thing as the code above.

```
int i;                      //declare the variable "i"

for ( i=1 ; i<=3 ; i++ )
{
    eyesGreen(100);         //turn eyes on
    delay(100);             //leave them on for 100ms
    eyesOff();              //turn eyes off
    delay(100);             //leave them off for 100ms
}
```

Wink_Ch11ForWhile_Ex01

"For" and "While"

Lesson 11

Let's have a look at the parts of the "for" and explain how they work. This section may seem a bit complicated at first. You may have to read it a few times but don't get hung up on the details. You'll understand how it works after a few examples.

```
for ( i=1 ; i<=3 ; i++ )          //'for' loop setup
{                                 //opening curley
  //code that runs each time
}                                 //closing curley
```

The "for" loop basically runs the code between its pair of curley braces while a certain condition is true. There are three important items inside the parenthasis that set up the "for" loop. Notice that each of these important items are separated by a ; semicolon.

1. The first part `i=1` is called the "initialization". The variable named `i` is going to count as we run the loop each time. We could name this variable anything, but it is customary to name it "`i`" when using "for" loops. We are going to use this variable `i` to count how many times we run the loop. This "initialization" sets the counting varible to some starting value.
2. The second part `i<=3` is the "condition" that will be tested ("is `i` less than or equal to three?"). Each time the loop runs, this condition is tested first. If the condition is true, then the code inside the curley braces is run (similar to how the "if" control structure works). If the condition is not true, then the "for" loop stops and the program moves on to whatever code is below the closing curley of the "for" loop. In this case, it asks the question "is it true that `i` is less than or equal to three?".
3. The third part `i++` tells the program what to do with the variable `i` each time the loop finishes running. If you write a variable name followed by two plus signs, this tells the program to add one to the variable. So in this example, we are telling the "for" loop to increase the value in the `i` variable by one each time it completes.

So working through the above example, when the "for" loop first begins to run, the value in the variable "`i`" is set to one. The loop then asks "is it true that `i` is less than or equal to three?". At this point `i` is equal to one, so the condition is true and the loop will run. The code inside the curley braces is then run. After this code runs, the value of `i` is automatically increased by one.

This continues until the loop has completed it's third time running. The first time it runs the loop `i` is equal to one. The second time, `i` is equal to two, and the third time, `i` is equal to three. As it completes this third time through the loop, `i` will be increased by one again so it now equals four.

When the "for" loop again tests the condition "is it true that `i` is less than or equal to three?" it will no longer be true, because four is not less than or equal to three.

I know it's a bit confusing, so the best way to really understand this is by doing a few examples.

Lets try an example where we print the value of i to the serial monitor window so you can see what happens as the program runs. This is similar to the above example except we've slowed it down a bit and added some Serial.print functions to tell us what is happening.

```
void loop(){

  int i;                    //declare the variable "i"

  for ( i=1 ; i<=3 ; i++ ){

    Serial.print("i = ");  //print lable to serial monitor
    Serial.println(i);     //print the present value of i

    eyesGreen(100);       //turn eyes on
    delay(100);           //leave them on for 100ms
    eyesOff();            //turn eyes off
    delay(900);          //leave them off for 900ms
  }      // closing curley of "for" loop

  Serial.println();         //print a blank line
  delay(2000);              //delay 2 seconds

} //end of loop()
```

} Declare "i" variable.

} Loop setup and opening curley

} Print present value of i to serial monitor

} Blink eyes

} For loop closing curley

} Print a blank line and wait two seconds before repeating.

Wink_Ch11ForWhile_Ex02

Load up this example and open your Serial Monitor window to observe what happens. You can see the first time through, i is equal to one. Then two, then three, then the fourth time the loop begins to run, i is no longer less than or equal to three (because it is now four). So the condition is no longer true and the "for" loop stops running and the program continues to the code below the "for" loop closing curley, which prints a blank line to your Serial Monitor window. The entire process begins again after the two second delay.

Using the counting variable "i" inside the loop...

As we see from the previous example, you can actually use the value of "i" iside the loop itself. This makes it possible to do interesting things. Let's try a challenge. Try to figure this one out on your own then check the example on the next page.

Can you think of a way to make Wink's eyes slowly go from dim to bright using a "for" loop?

Dim to bright example...

```
void loop(){

  int i;                  //declare the variable "i"

  for ( i=1 ; i<=250 ; i++ ){

    eyesGreen(i);         //set eye brightness based on "i"
    delay(4);             //change this to control fade speed
  }

} //end of loop()
```

} Set eye brightness based on "i"

Wink_Ch11ForWhile_Ex03

Load up this example and see what happens. Each time the "for" loop begins, "i" is set to one. Each time the loop runs, we use the variable "i" inside the eyesGreen() function instead of a written number. As "i" increases as the loop runs, the eyes get brighter.

Counting backward...

Sometimes it is useful to count "i" backward by decreasing its value each time instead of increasing it as we have done so far. Like this...

```
void loop(){

  int i;                  //declare the variable "i"

  for ( i=250 ; i>1 ; i-- ){

    eyesPurple(i);        //set eye brightness based on "i"
    delay(10);            //change this to control fade speed
  }

  delay(2000);            //delay before doing it again

} //end of loop()
```

} By using i-- we count backward

Wink_Ch11ForWhile_Ex04

In this example we initialize i to a higher value. In the test condition, we ask if i is greater than one. Each time the loop runs, i-- causes i to be decreased by one each time the loop finishes. Eventually i will decrease far enough that it is no longer greater than one and the "for" loop will finish and move on to the next line of code below the closing curley of the "for" loop.

More about this ++ / -- thing...

So we introduced a new concept in this lesson that we haven't yet covered. We left it out until now so you weren't confused earlier. You see we have been using `i++` and `i--` to tell the "for" loop what to do.

We have also explained that by adding two plus signs right behind a variable name will cause the program to automatically add one to it. The opposite is true with two minus signs, which will automatically subtract one from it.

That is to say...

```
int birds = 1;    //"birds" declared and assigned value of 1

birds = birds + 1;  //add 1 to birds. birds now equals 2

birds++;  //automatically add 1 to birds. birds now equals 3

birds--;  //automatically subtract 1 from birds. birds now 2
```

Does that mean you can do something other than just add or subtract one each time you go through the "for" loop? It sure does. You can make the "for" loop count by twenty five (or any other number for that matter) each time like this...

```
void loop(){

  int i;                    //declare the variable "i"

  for ( i=0 ; i<=250 ; i = i+25 ){

    eyesPurple(i);          //set eye brightness based on "i"
    delay(100);             //change this to control fade speed
  }

  offEyes();                //turn off eyes
  delay(2000);              //delay before doing it again

} //end of loop()
```

} You can use any assignment equation to increase or decrease the "i" counter

Wink_Ch11ForWhile_Ex05

You can do lots of other things with "for" loops that are more complicated, but these examples cover the basics. Write some "for" loops on your own to do something interesting. Can you use a "for" loop to slowly increase the speed of Wink's motors?

"for" loop review...

Now that you've seen it all a few times, let's review in case it's still not quite clear.

```
for ( i=1 ; i<=10 ; i++ )
```

This part runs one time at the very start of the "for" loop process. It is usually used to set your counting variable to some starting number. You can set this starting value to anything you want as long as the value can be stored in the type of variable you have declared. (We have been declaring this as an "int" type - go back and review the Variables lesson if needed).

```
for ( i=1 ; i<=10 ; i++ )
```

This part is the "condition" that will be evaluated just before the code inside the "for" loop is run. It works just like the "condition" used in an "if" control structure as learned in a previous lesson. If this condition is true, the code inside the curley braces { } of the "for" loop will be run. If this condition is not true, the code inside the curley braces is not run, the "for" loop stops running and the program moves on to the first line of code after the closing curley } of the "for" loop.

```
for ( i=1 ; i<=10 ; i++ )
```

This part runs at the end of the "for" loop. It automatically runs after the code inside the curley braces has completed running.

After this part runs, the "condition" part will be evaluated again. If that condition is still true, the code inside the curley braces will run again. If that condition is no longer true, the "for" loop stops.

Doing things "while" something is true...

Remember back when we learned the "if" control structure? We used "if" to do something "if" a condition was true. The "while" control structure works exactly the same, except that it does something and continues doing something "while" a condition is true.

This is useful in many cases. For example, you may want a robot to do something and continue doing the same thing over and over while you press and hold a button.

Let's try an example...

```
void loop(){

  while ( buttonPressed() ){    //do this BTN is pressed
    eyesPink(50);
    delay(75);
    eyesOff();
    delay(75);
  }

} //end of loop()
```

Wink_Ch11ForWhile_Ex06

} The function `buttonPressed()` will be true if Wink's BTN button is pressed (the button on the right side of the robot)

You can use "while" the same as using "if". In the past we have used the condition to compare two values, but you can also use a condition to determine if a function itself is "true". We'll learn functions soon enough, but know that a function like buttonPressed() can be made to be "false" if a button is not pressed, and "true" if a button is being pressed.

In this case we use the "while" control structure together with the buttonPressed() function to do something and continue doing something as long as the button is being pressed.

Using the "NOT" operator...

As long as we're using a function to determine if something is true or not, let's talk about the "not" operator. The "not" operator is an exclamation ! mark. What if we want Wink to only do when his button is "not" being pressed?

We can place an exclamation mark in front of the buttonPressed() function. This will make the condition true if the button is NOT being pressed. The NOT operator can be used in "if" control structures the same way.

Hopefully you are starting to realize that the C language can be used in many ways. There are usually many ways to accomplish the same task. Try to use a NOT operator yourself then check the next example.

Here is an example of using the NOT operator...

```
void loop(){

  while ( !buttonPressed() ){   //while NOT buttonPressed()
    eyesPink(50);
    delay(125);
    eyesOff();
    delay(125);
  }

} //end of loop()
```

} The NOT ! operator before the condition

Wink_Ch11ForWhile_Ex07

Making "while" work like "for"...

Now we're getting really creative. Let's consider one more example to show you how flexible the C language can be. Can you think of a way to make "while" behave like "for"? Can we re-write the previous "for" example as a "while"? Try it on your own if you like.

Here's what I came up with...

```
int i;                //declare the variable "i"

void loop(){
  i = 1;              // set i equal to 1

  while (i<=250){
    eyesGreen(i);     //set eye brightness based on "i"
    delay(4);         //change this to control fade speed
    i = i + 1;        // increase i by one
  }
} //end of loop()
```

} Declare variable "i"

} Set i equal to 1 before "while" starts

} Use "while" instead of "for"

} Set eye brightness based on "i"

} Increase i. You could also use i++ here

Wink_Ch11ForWhile_Ex08

If we make "while" run based on i, then increment or somehow change i inside the loop, it can eventually make the "while" condition no longer true. This code should do exactly the same thing as the earlier example 3 of this lesson, just using a different method.

You can write your code in whatever way makes the most logical sense to you. There are usually many ways to do the same thing.

You now understand "if", "for", and "while". Using these three control structures you can make your program do all kinds of different things based on just about anything.

Using while(1)...

Let's take the idea of the "while" control structure a step further.

We know that "while" and "if" will run the code inside their curley braces if the condition is "true". Normally this condition is comparing at least two things together, like whether a variable is greater than a certain number.

You can also use the condition to look at just a single thing, rather than comparing two things.

When looking at just one item inside the condition, it is said that something is "true" if it is not zero. Inside "if" and "while", when the condition is looking at a single item, if it is not zero the condition is considered to be "true". Anything that is zero is considered to be "false".

We did this without realizing it in the previuos examples where we used buttonPressed() by itself as the condition. The function buttonPressed() goes off and runs a bit of code. After that code completes, the function "returns" back with either a zero or a one. If the button was being pressed, it returns with a one, and if the button was not being pressed, it returns with a zero.

When looking over code examples, you will eventually see this...

```
while(1)
{
  // some code
}
```

} Using while(1)

Let's think about how this works. The "while" looks at the condition, which is one, which is "true", so whatever is inside the curley braces runs. Then "while" looks at the condition agin, which of course is still true because the one doesn't change.

This is a common way to get something to run forever (or at least until the power is turned off and back on, or the reset button is pressed which resets the processor and starts it running code from the start of the program again). There are many situations where you may want something to run forever.

For example, maybe your program is controlling several large and powerful motors. If that program reads a sensor that indicates one of the motors is over loaded or there is some other problem, the code may run a while(1) loop that turns off power to everything and stops the motors. By making the program get stuck inside an endless "while" loop, you can be sure a person must get involved to correct the problem and intentionally re-start the process.

If you study the Wink demo behaviors, you'll see we use this method once a behavior is started so it keeps running over and over.

Using "break"...

If you ever want to break out of a "while" loop, you can use the "break" statement by itself. Like this...

```
while(1)
{
  break;          //breaks out of the while loop
}
```

} The "break" statement causes the program to immediately exit the "while" loop and run the first line of code after the closing curley of the "while".

See how it works in this example...

```
void loop(){

  eyesBlue(100);
  delay(2000);

  while(1){
    eyesPink(50);
    delay(200);
    eyesOff();
    delay(200);

    if(buttonPressed()){
      break;              //break exits while(1)
    }
  }

} //end of loop()
```

Wink_Ch11ForWhile_Ex09

} Eyes blue for 2 seconds so we can see the start of the process.

} Blink sequence

} Break out of while(1) if button is being pressed

Each time the while(1) loop runs, it checks if the button is being pressed and if it is, the single line `break;` is run, which causes the program to break out of the while loop.