```
int main() {
   return 0;
}
```

In the previous tutorial, we have written the **bare minimum PROGRAM in C-Language** (the code above)

When the micro-controller is Powered-Up, the CPU will execute all the instruction codes within the main() function body within the curly bracket {} pair. Ideally we must put whatever instruction code that we want the CPU to perform, before the "return 0;" instruction code.

Once the CPU executes the "return" code, function ends

When the main() function ends, our PROGRAM is "dead" and our micro-controller does not have any more instruction codes to run, it is now "brain dead". We do not want that to happen. We want to "live forever"

"SUPER LOOP"

Arduino IDE|Save PROGRAM as: **c_super_loop**
Enter codes below and upload.

```
int main() {
   while(1)
   {
      // The codes within the "while" curly bracket { } pair
      // will be repeated forever
   }
   return 0; // our main() function will never reach here
}
```

In this PROGRAM, the "return 0;" instruction code inside our main() function will never be executed by the CPU. Meaning our main() function will never end, our PROGRAM 'lives forever"

The "instuction codes" within the **"while loop"** body section will be repeated forever because of the **"while(1)"** code

The forever "while loop" in the main() function is known as

"SUPER LOOP"

In **Regular Computers**/Smartphone type of micro-controllers, we keep the CPU alive by using an **"Operating System PROGRAM"**. In **smaller micro-controllers** we use "SUPER LOOP" or another technique called **"RTOS – Real Time Operating System"**, where the "RTOS" allows us to run "multiple SUPER LOOP" inside PROGRAM. However, we will need more memory and processing power to use the RTOS. We will just stick to "SUPER LOOP" for now, **"SUPER LOOP"** is good enough for most PROGRAM used in micro-controllers

C-Language: programmers comment line "//"

The text after the "//" Symbol in our C-Language PROGRAM are "programmers comment line". They will not be included in the final PROGRAM. We put them in our PROGRAM Source Code to explain things

C-Language: while loop

There are 3 Parts in the "while loop"
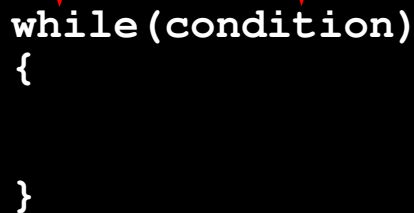
Part1: "while" Keyword
Part2: condition – within a bracket ( ) pair
Part3: body – "instruction codes" within a curly bracket { } pair

Part2: condition
– placed within the bracket ( ) pair
– value of 0 or NOT 0
– can be variable name or constant name which contains value of 0 or NOT 0
– can be an "expression" of a logic operation resulting in value of 0 or NOT 0
– can be a function returning a value of 0 or NOT 0

Part1: "while" Keyword

Part3: body
– contains "instruction codes"
– within a curly bracket { } pair

```
while(condition)
{


}
```

"while loop" Starts with **"condition evaluation:"**

– **when the condition is 0, "instruction codes" within the body will NOT be executed.** The PROGRAM continues with the "instruction codes" after the body closing curly bracket "}"

– **when the condition is NOT 0, "instruction codes" within the body will be executed.** After the last line of "instruction codes" in the body, PROGRAM will run the **"condition evaluation:"** again, just like before

Example:

```
while(1) {
   // This will run forever, condition has a fixed value 1, which is NOT 0
}
```

```
int cond_value = 1;
while(cond_value) {
   // This will run forever, as long as "cond_value" variable does not turn 0
}
```

```
int cond_value = 1;
while(cond_val == 1) {
   // This will run forever, as long as the logic "cond_value == 1" is true
   // true = 1, false = 0
}
```

```
int get_value(){return 1;}

while( get_value() ) {
   // This will run forever, as long as get_value() does not return 0
}
```

– We can forced an immediate **"exit from while loop"** by coding **break;**
– We can forced an immediate **"condition evaluation:"** by coding **continue;**

Step 1/3:
Start with a skeleton "while loop" with a default condition

```
while(0){}
```

Step 2/3:
We want to make a "SUPER LOOP", we need to repeat running the "instuction codes" in the "while loop" body "forever". We put a **non 0** for the "while loop" **condition** and make sure that condition will never change

```
while(1){}
```

At this point, we already have a functional "while loop" for the main() function "SUPER LOOP"

Step 3/3:
Lets put something in the "while loop" body, so that we will have something to "see"

```
while(1) {
  PORTB = PORTB ^ (1<<PB5);
  _delay_ms(250);
}
```

We should be coding the "while loop" inside our bare_minimum main() function directly. We do it seperately here, so that we can see how it is coded, step by step

Below, is our completed "SUPER LOOP" that lives forever (turning LED ON and OFF forever)

Arduino IDE|Save PROGRAM as: **c_super_loop_blink**
Enter codes below and upload.

```
#define F_CPU 16000000UL
#include <avr/delay.h>
#include <avr/io.h>
int main() {
  DDRB = DDRB | (1<<DDB5);
  while(1) {
    PORTB = PORTB ^ (1<<PB5);
    _delay_ms(250);
  }
  return 0;
}
```

Watch the LED on the Arduino Uno Board

The instruction codes inside the "while loop" body will be repeated forever because the "condition" for the "while loop" is always 1 (non-0)

#define F_CPU 16000000UL
#include <avr/delay.h>
– The two lines of codes above works together, they are required when we use the _delay_ms() function

#include <avr/io.h>
– The line of code above, is required when we use "DDRB","DDB5","PORTB" or "PB5" in our PROGRAM

Arduino IDE|Save PROGRAM as: **c_super_loop_blink_function**
Enter codes below and upload. Watch the LED on the Arduino Uno board

```c
#define F_CPU 16000000UL
#include <avr/delay.h>
#include <avr/io.h>
int main() {
  setup_run_once(); // move the codes previously here into a function
  while(1) {
    loop_run_forever(); // move the codes previously here into a function
  }
  return 0; // our main() function will never reach here
}

void setup_run_once() {
  DDRB = DDRB | (1<<DDB5);
}

void loop_run_forever() {
  PORTB = PORTB ^ (1<<PB5);
  _delay_ms(250);
}
```

These are exactly the same

Arduino IDE|Save PROGRAM as: **c_super_loop_blink_arduino**
Enter codes below and upload. Watch the LED on the Arduino Uno board

```c
void setup() {
  DDRB = DDRB | (1<<DDB5);
}

void loop() {
  PORTB = PORTB ^ (1<<PB5);
  _delay_ms(250);
}
```

The setup() and loop() functions in our regular Arduino Libaries PROGRAM are actually components of a "SUPER LOOP" BUT where is the main() function ?

When we make PROGRAM using the **setup()** and **loop()** with the Arduino IDE Software, **the main() function has already been coded for us**. It is just being hidden from us by the Arduino IDE Software. The main() function is kept in a .cpp file in ther Arduino IDE installation folder:

<install folder>/hardware/arduino/avr/cores/arduino/main.cpp

```c
int main(void)
{
  init();
  initVariant();
  #if defined(USBCON)
    USBDevice.attach();
  #endif
  setup();
  for (;;) {
    loop();
    if (serialEventRun) serialEventRun();
  }
  return 0;
}
```

< codes extracted from main.cpp
This main() function is doing more things than ours

The hidden main() function provided by the Arduino IDE Software uses the "for loop" instead of the "while loop" in our example, both have the same effects

setup() and **loop()** function run from the main() function

From now onwards, we will be writing our PROGRAM using the C-Language with Arduino Libraries and the Arduino "SUPER LOOP"

We will not see the main() function anymore on our Arduino IDE Software. We will only see and code inside the **setup()** and **loop()** function

This style of coding is very much easier for learning process, as alot of details are already taken care for us by the Arduino Libraries

However, once a while we will still dive into the details, especially when it is something important

When we start writing a new PROGRAM.

We will start from the "bare minimum skeleton" code below, then we expand from there

```
void setup(){}
void loop(){}
```