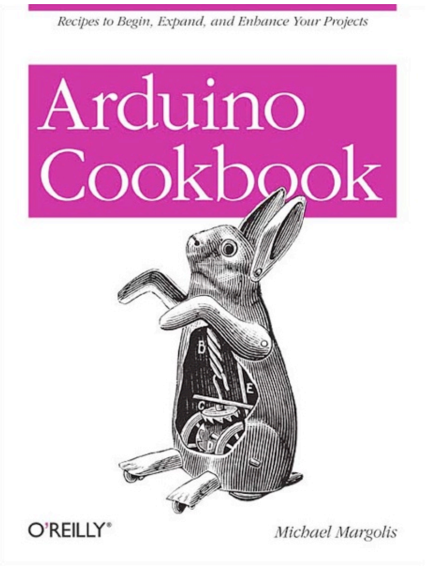
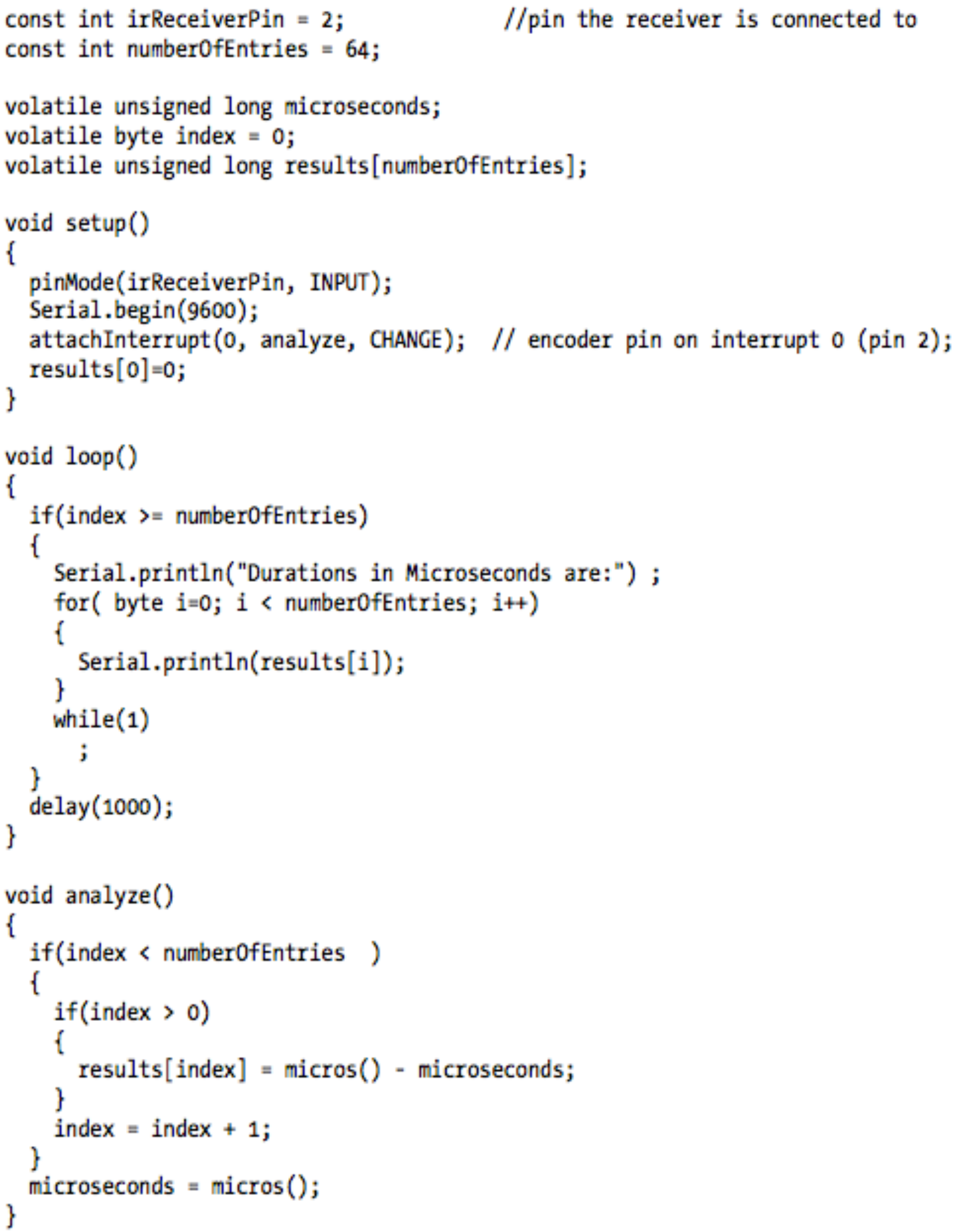
Interrupts in Arduino

Reference: Arduino Cookbook" (1st ed.) by Michael Margolis.

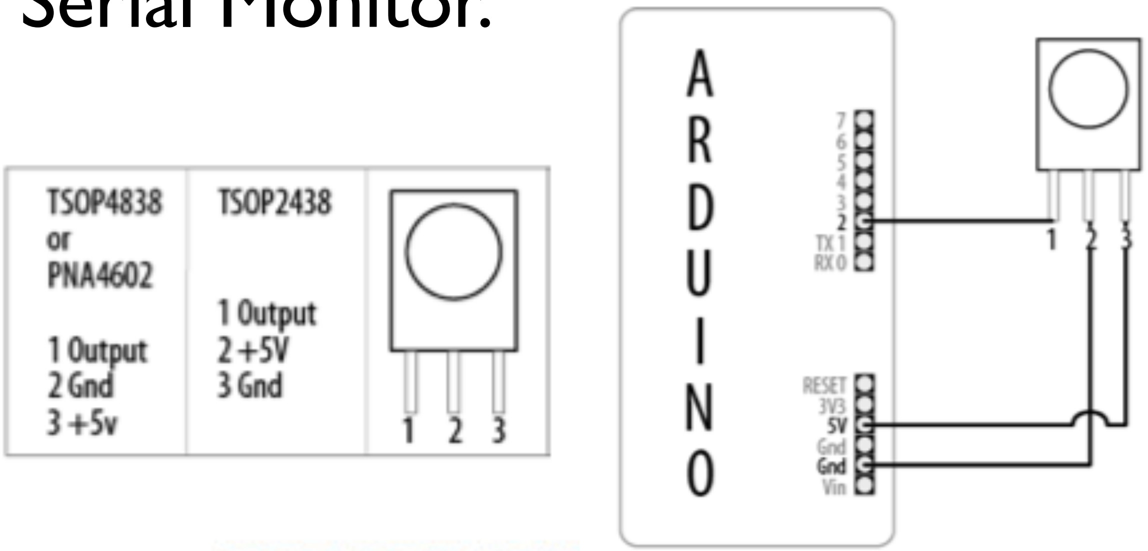


Sample Code

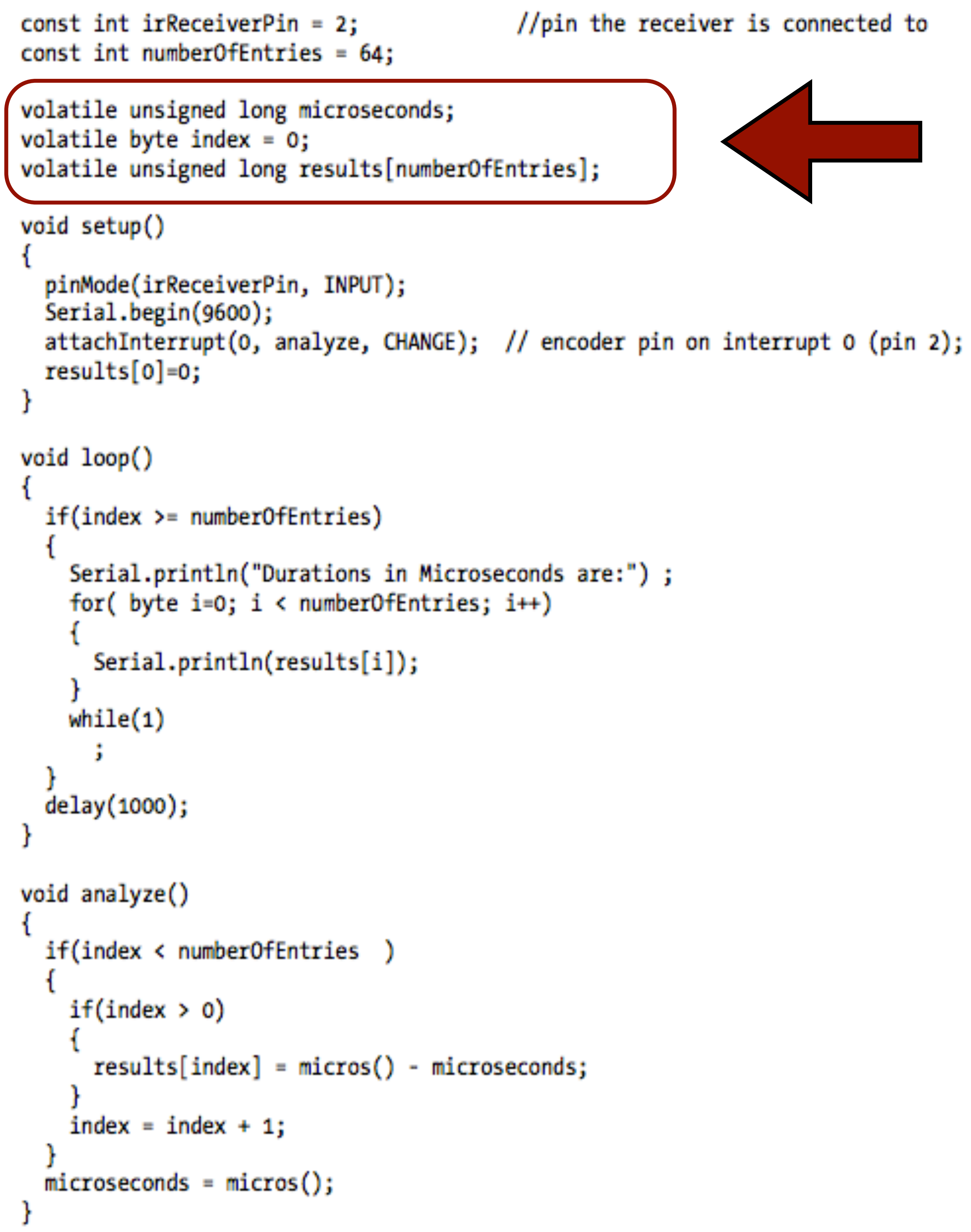


```
const int irReceiverPin = 2;            //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- You have an IR detector connected to pin 2.
- This program monitors pulses on pin 2 and stores the duration of each pulse in an array.
- When the array has been filled each duration is displayed on the Serial Monitor.



Volatile Keyword



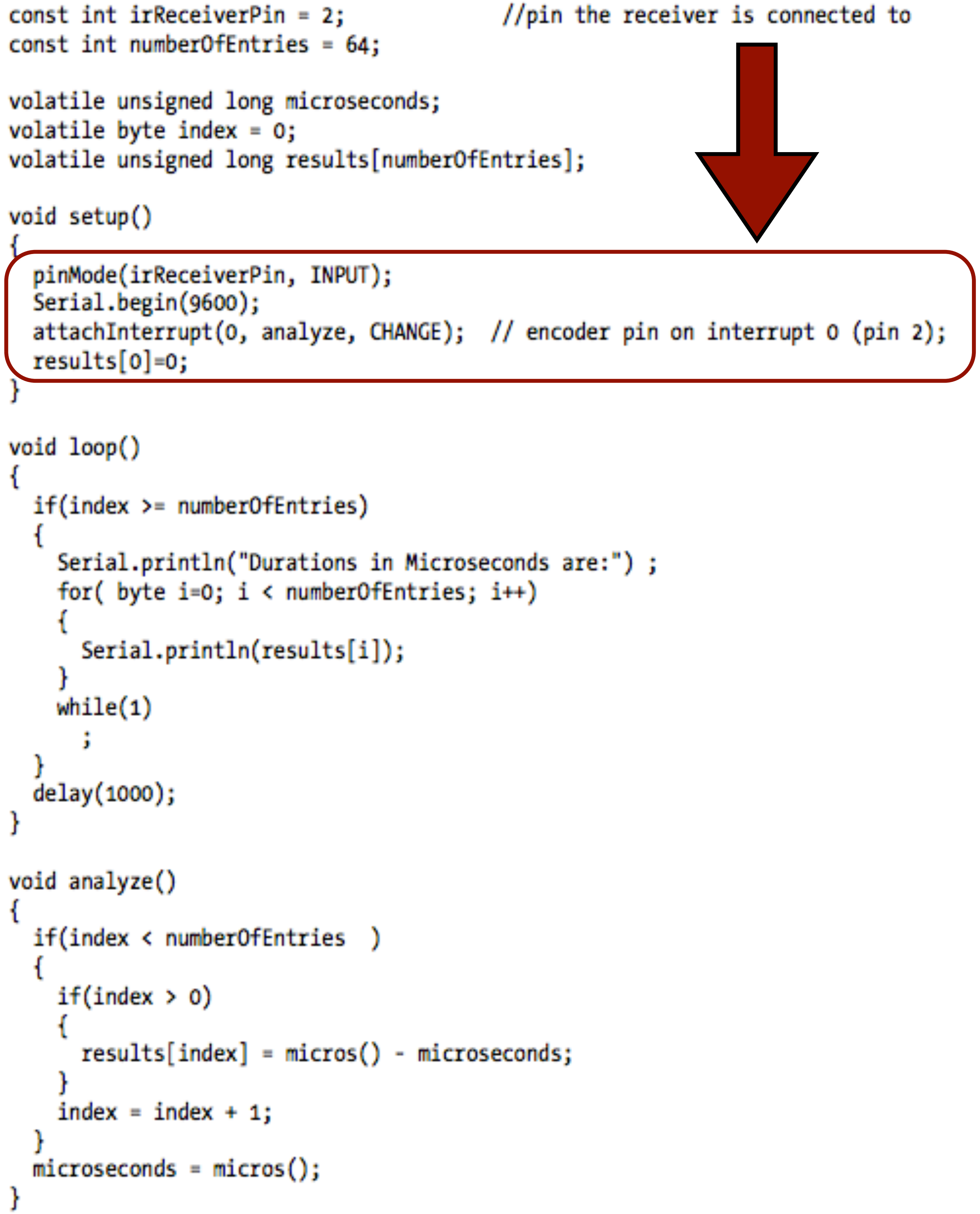
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- Its a **variable qualifier**; it is used before the datatype of a variable, to modify the way in which the compiler and subsequent program treats the variable.

- Compiler will load the variable from RAM and not from a storage register.

- Under certain conditions, such as through interrupts, the value for a variable stored in registers can be inaccurate.

attachInterrupt function



```
const int irReceiverPin = 2;          //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

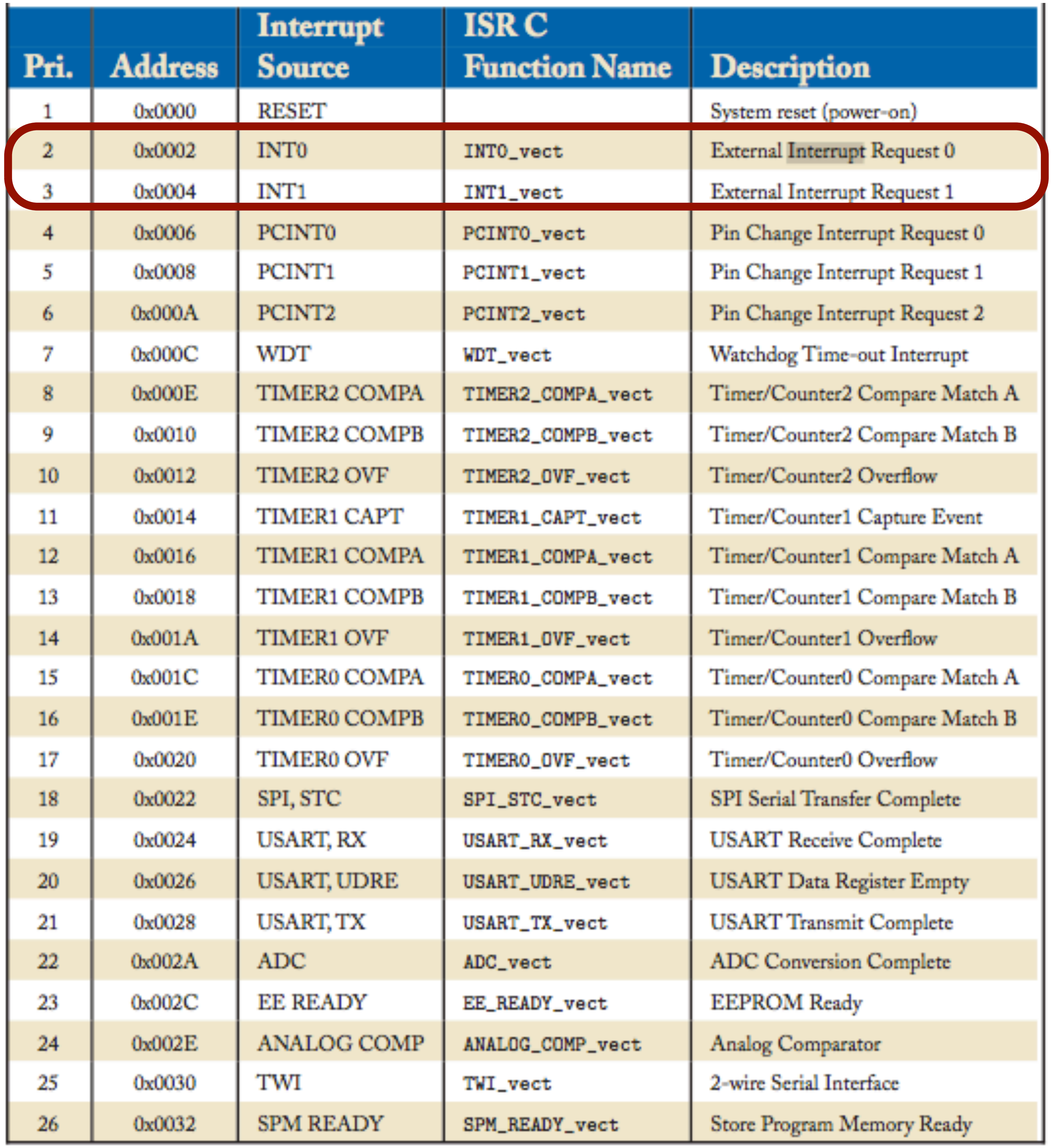
- The attachInterrupt(0, analyze, CHANGE); call enables the program to handle interrupts.

- The first number in the call specifies which interrupt to initialize.

- On a standard Arduino board (such as UNO), two interrupts are available: number 0, which uses pin 2, and number 1 on pin 3.

- Interrupt 0 and interrupt 1 have the same priorities (with Wiring).

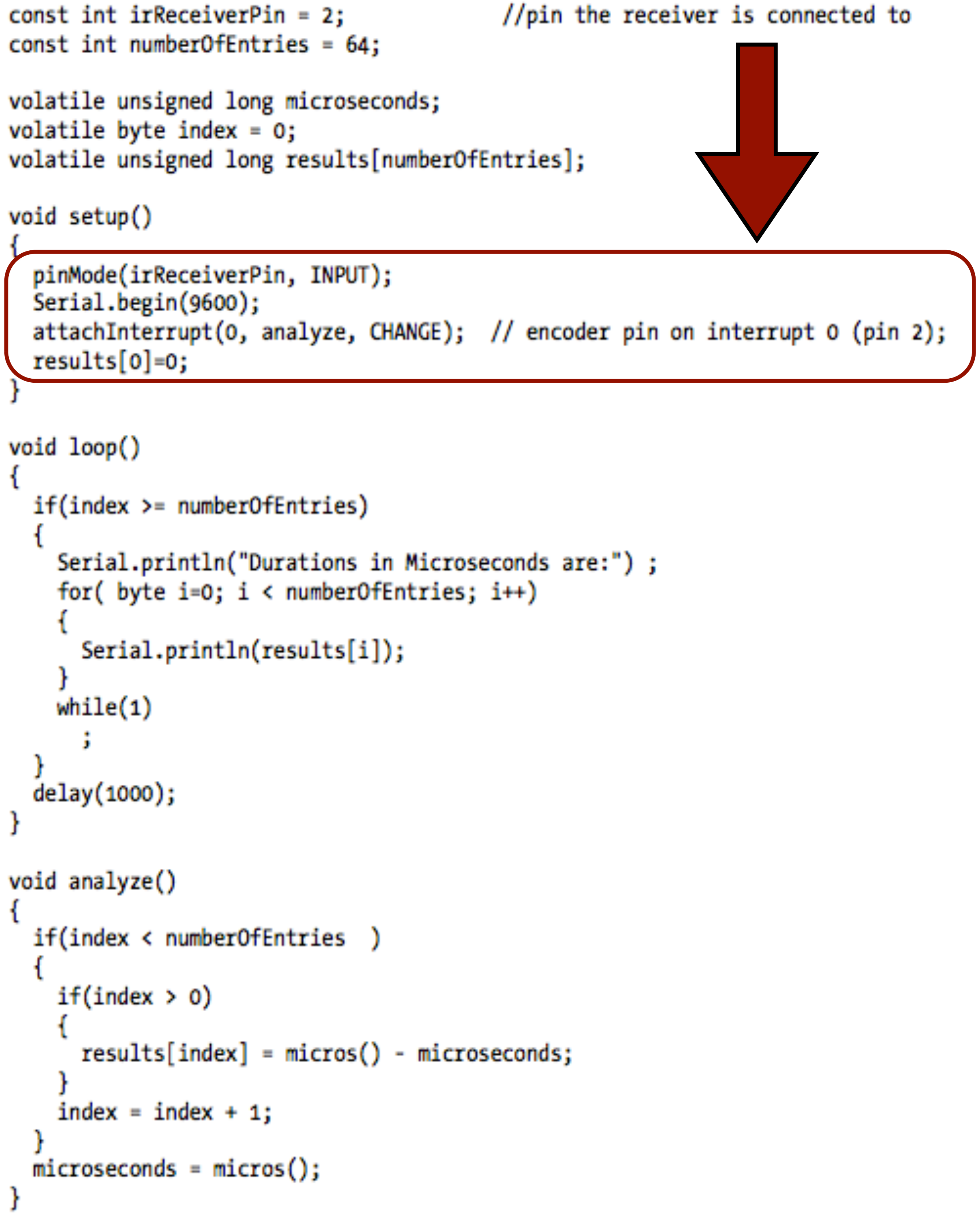
A lot more interrupts



Pri.	Address	Interrupt Source	ISR C Function Name	Description
1	0x0000	RESET		System reset (power-on)
2	0x0002	INT0	INT0_vect	External Interrupt Request 0
3	0x0004	INT1	INT1_vect	External Interrupt Request 1
4	0x0006	PCINT0	PCINT0_vect	Pin Change Interrupt Request 0
5	0x0008	PCINT1	PCINT1_vect	Pin Change Interrupt Request 1
6	0x000A	PCINT2	PCINT2_vect	Pin Change Interrupt Request 2
7	0x000C	WDT	WDT_vect	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	TIMER2_COMPA_vect	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	TIMER2_COMPB_vect	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	TIMER2_OVF_vect	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	TIMER1_COMPB_vect	Timer/Counter1 Compare Match B
14	0x001A	TIMER1 OVF	TIMER1_OVF_vect	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	TIMER0_COMPA_vect	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	TIMER0_COMPB_vect	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	TIMER0_OVF_vect	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI_STC_vect	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART_RX_vect	USART Receive Complete
20	0x0026	USART, UDRE	USART_UDRE_vect	USART Data Register Empty
21	0x0028	USART, TX	USART_TX_vect	USART Transmit Complete
22	0x002A	ADC	ADC_vect	ADC Conversion Complete
23	0x002C	EE READY	EE_READY_vect	EEPROM Ready
24	0x002E	ANALOG COMP	ANALOG_COMP_vect	Analog Comparator
25	0x0030	TWI	TWI_vect	2-wire Serial Interface
26	0x0032	SPM READY	SPM_READY_vect	Store Program Memory Ready

- Yes, there are a lot of other interrupts, but they are internal.
- We will user the timer interrupts very soon.
- Unsurprisingly the RESET interrupt has the highest priority.
- This table is from Russell's book (listed on syllabus).

attachInterrupt function

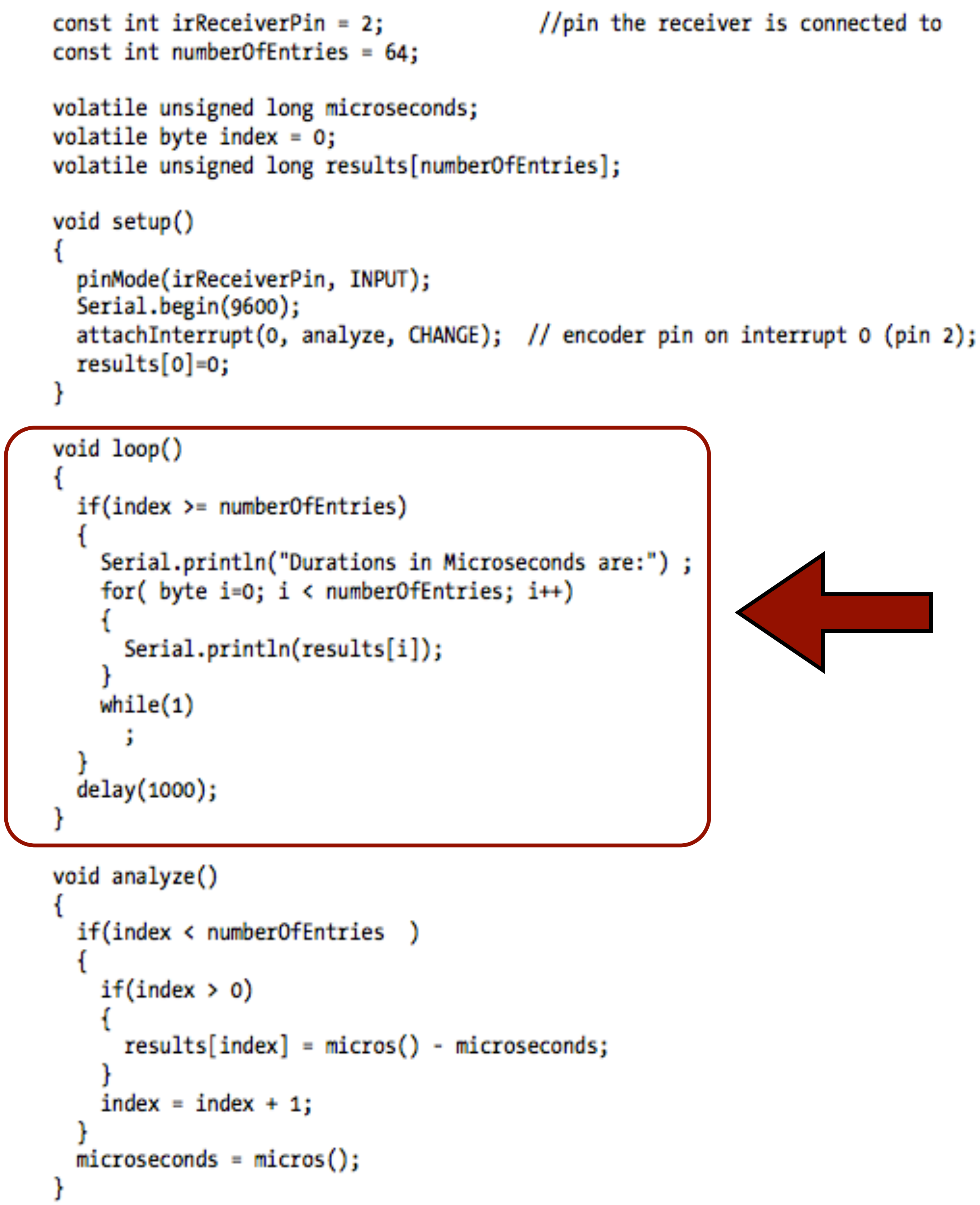


```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries   )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- The second parameter specifies what function to call (**interrupt handler**) when the interrupt event happens.

- The final parameter specifies what should trigger the interrupt:
  - CHANGE: whenever the pin level changes (low to high or high to low).
  - LOW: when the pin is low.
  - RISING: when the pin goes from low to high.
  - FALLING: when the pin goes from high to low.

Main loop



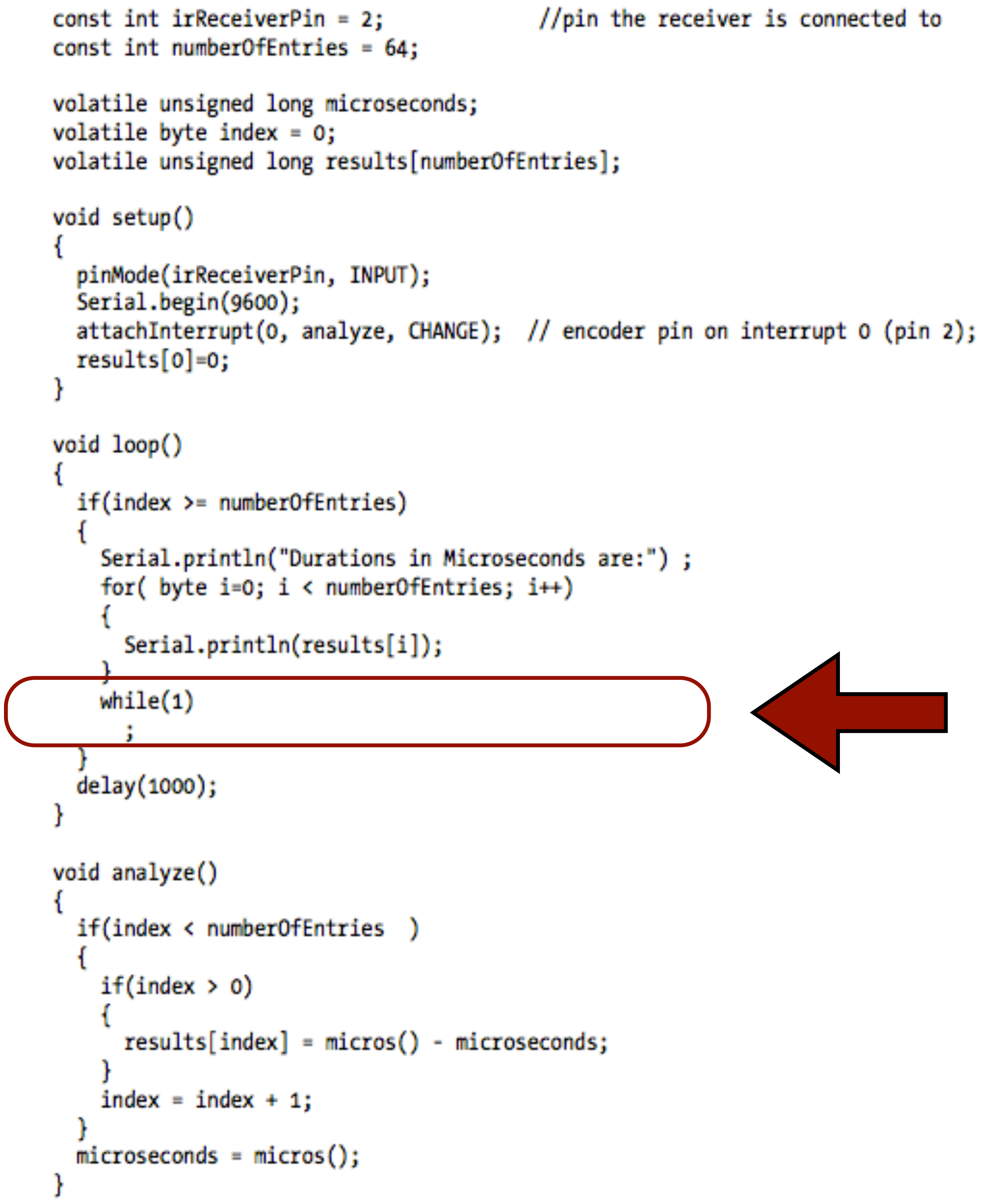
```
const int irReceiverPin = 2;            //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- The main loop just checks the *index* variable to see if all the entries have been set by the interrupt handler.

- ...And it will print the contents of the array *results* only once.

- Nothing in loop changes the value of index. The index is changed inside the analyze function when the interrupt condition occurs.

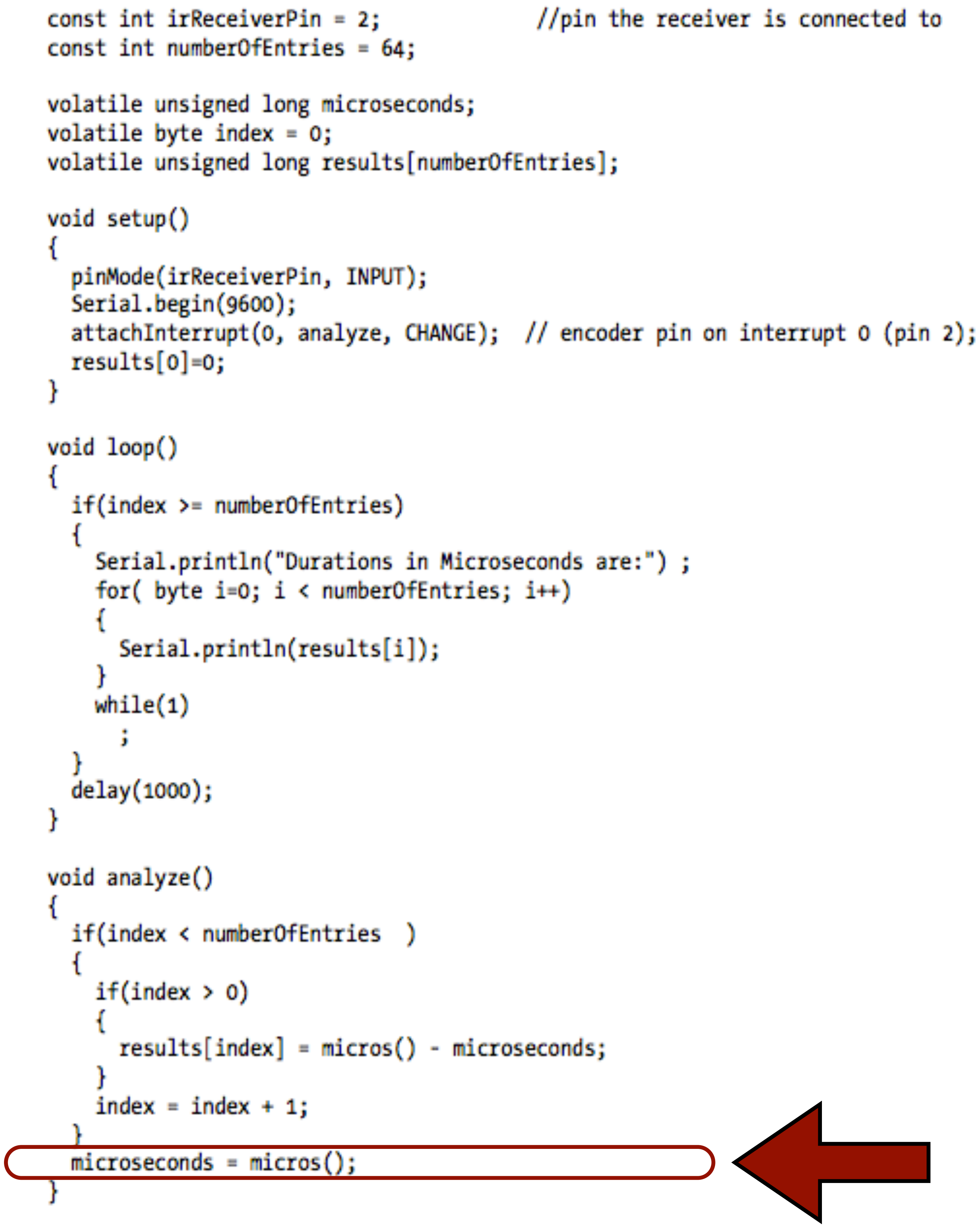
Termination condition



```
const int irReceiverPin = 2;            //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- The code stays in the while loop at the end of the inner block, so you need to reset the board when you want to do another run.

micros() function



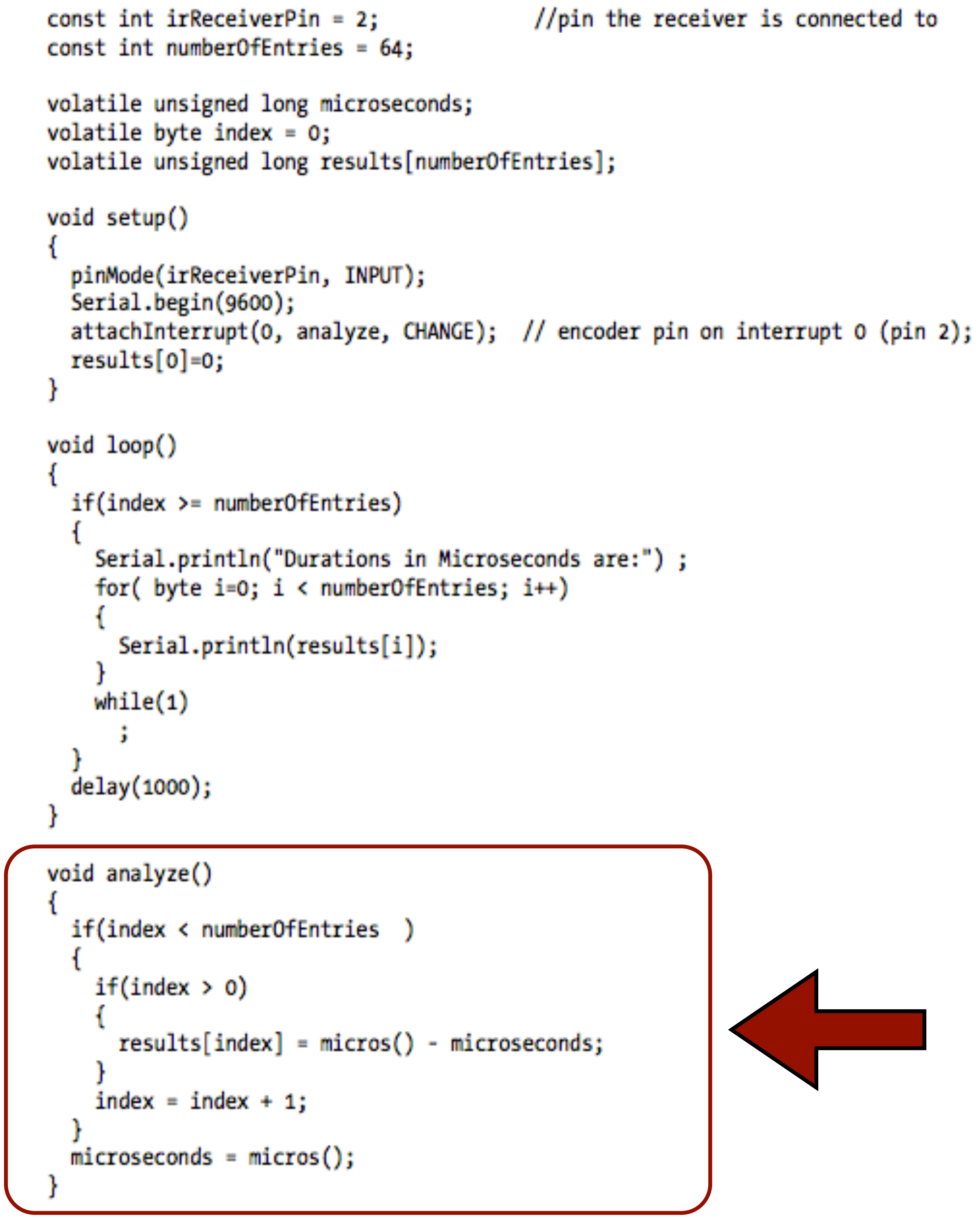
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- The micros() function returns the number of micro-seconds since the Arduino began running the current program.

- This number will overflow after This number will overflow (go back to zero), after approximately 70 minutes.

- On 16 MHz Arduino boards (e.g. UNO), this function has a resolution of four microseconds (i.e. the value returned is always a multiple of four).

analyze() function



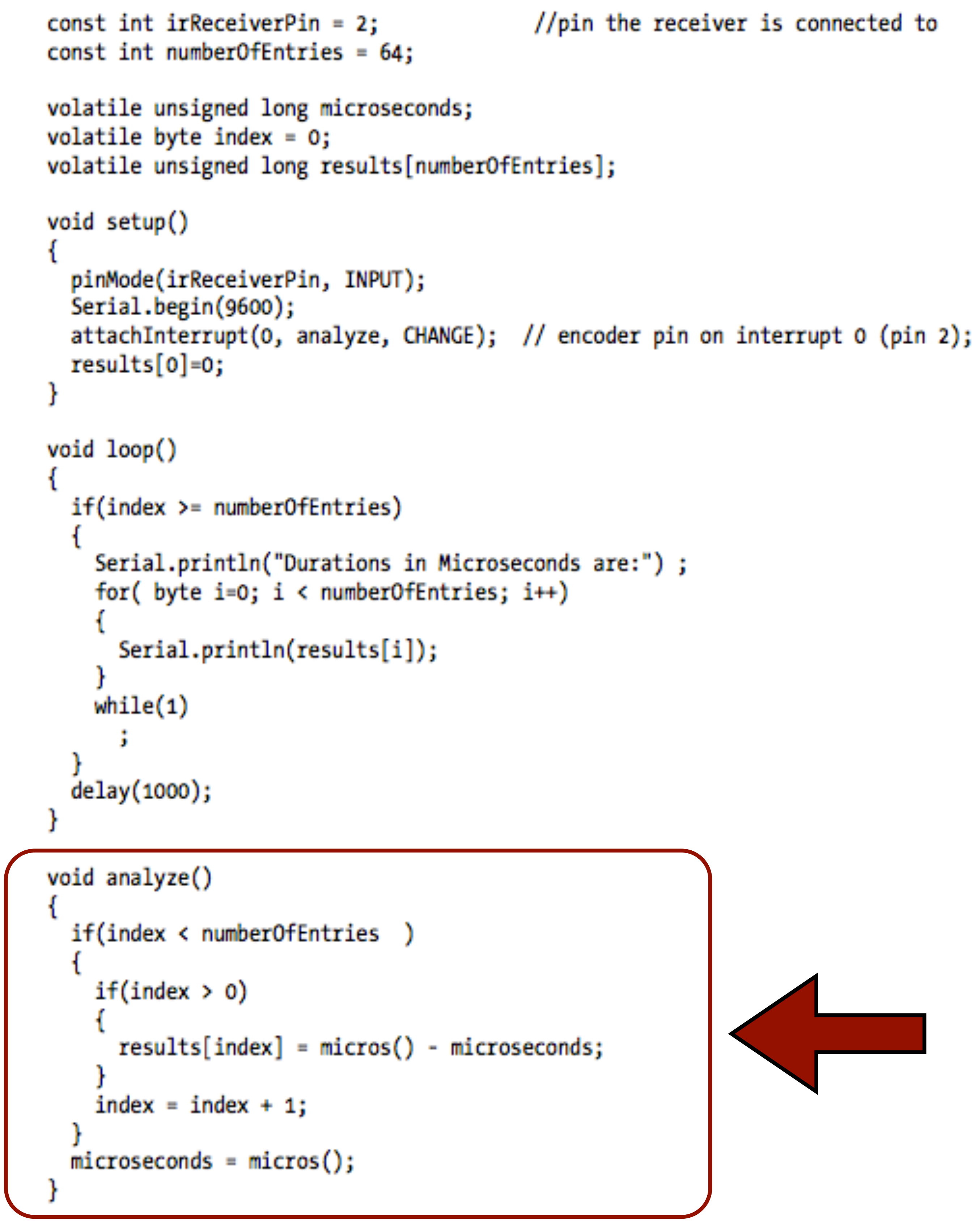
```
const int irReceiverPin = 2;            //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries   )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- The index value is used to store the time since the last state change into the next slot in the results array.

- The time is calculated by subtracting the last time the state changed from the current time in microseconds.

- The current time is then saved as the last time a change happened.

Changing variables



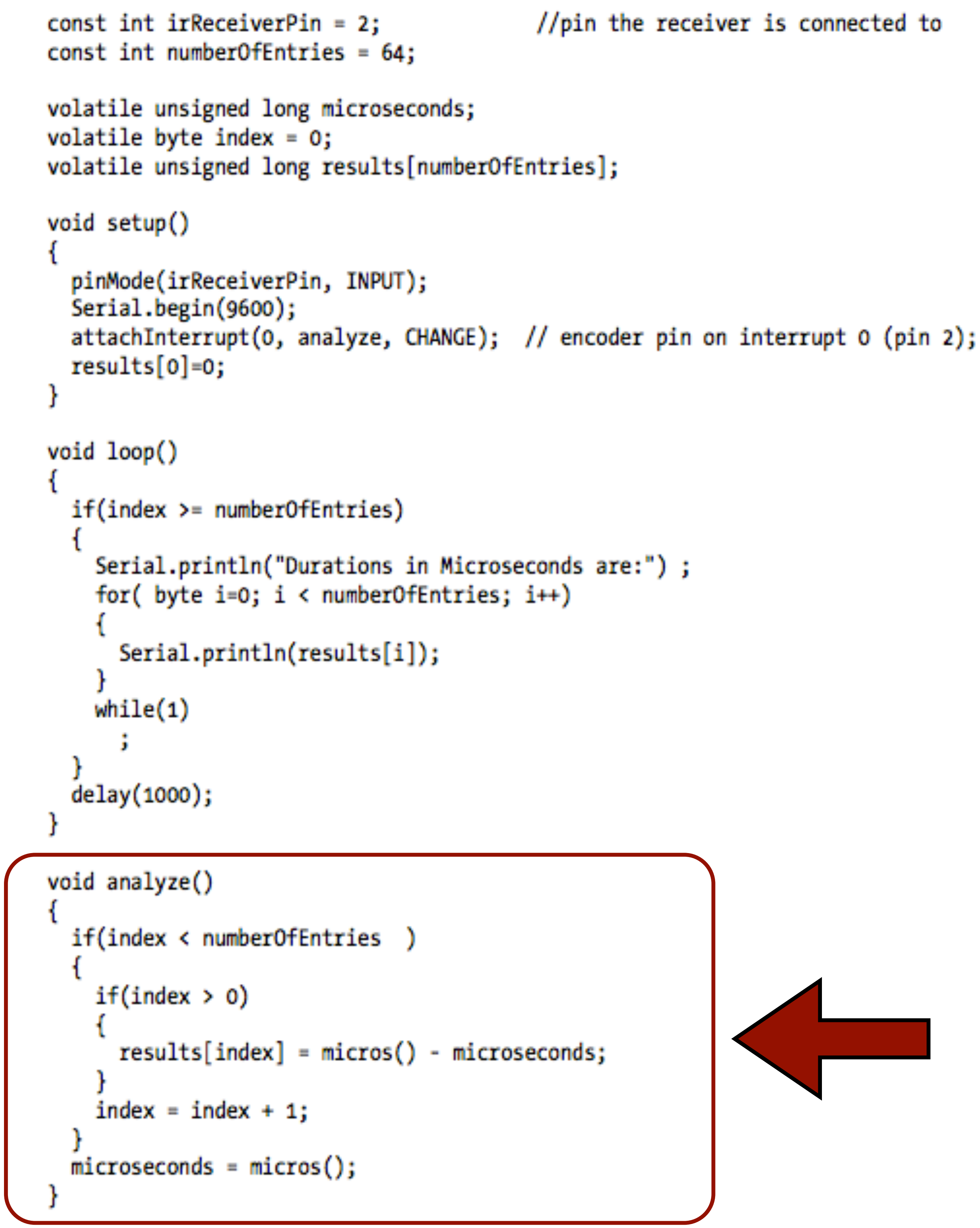
```
const int irReceiverPin = 2;            //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- The variables that are changed in an interrupt function are declared as volatile.

- This lets the compiler know that the values could change at any time (by an interrupt handler).

- Without using the volatile keyword, the compiler would think these variables are not being changed by any code getting called and would replace these variables with constant values.

What's the code doing?



```
const int irReceiverPin = 2;            //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE);  // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:") ;
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
      ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries  )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

- Each time an interrupt is triggered, index is incremented and the current time is saved.

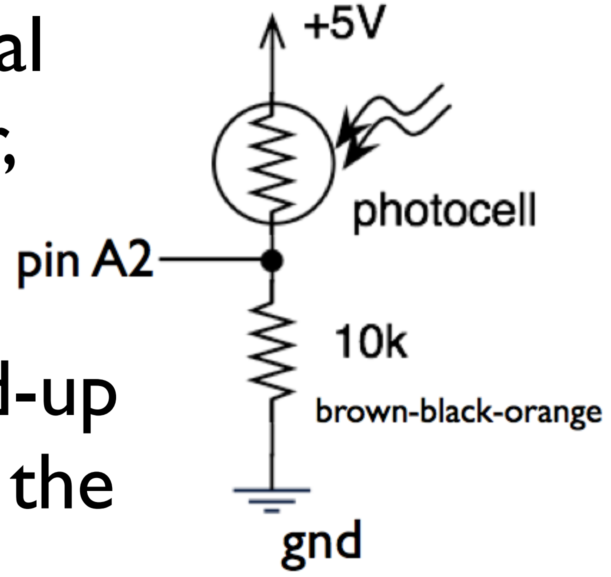
- The time difference is calculated and saved in the array (except for the first time the interrupt is triggered, when index is 0).

- When the maximum number of entries has occurred, the inner block in loop runs, and it prints out all the values to the serial port.

Another sample code

```
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}

void blink()
{
  state = !state;
}
```

- Attach something that will trigger an interrupt in digital pin #2 (e.g. resistive sensor, button, ...)
- Make sure its either pulled-up or pull-down, as shown on the right.



- Attach a LED to digital pin 13. Whenever, the interrupt is triggered (through pin #2 changes), the LED will change state.

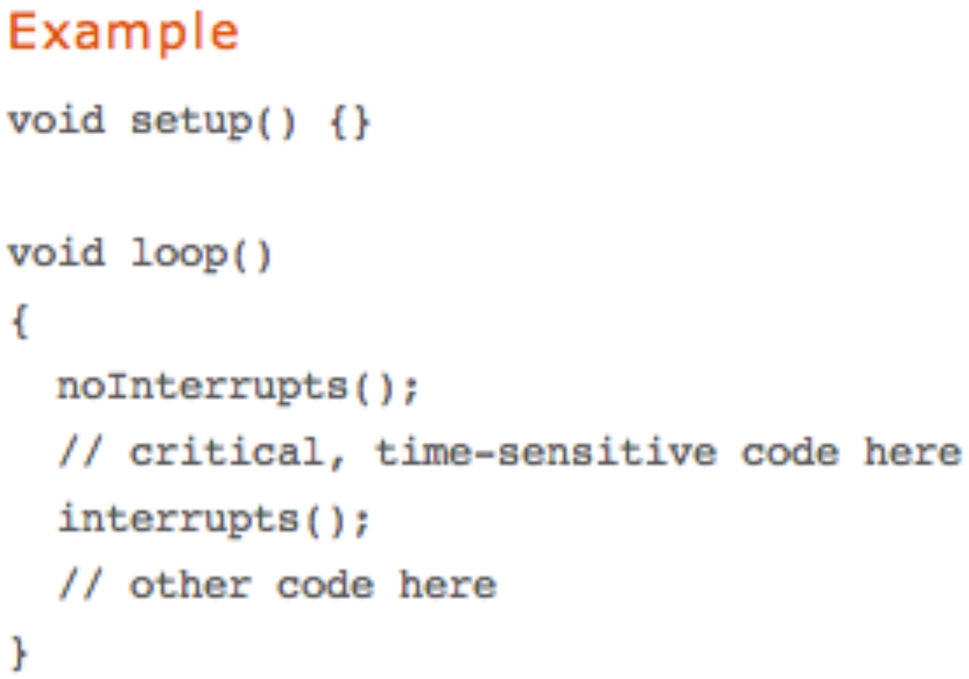
Arduino wiring C limitations

```
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}

void blink()
{
  state = !state;
}
```

- Inside the interrupt function, delay() won't work
- Values returned by the millis() and micros() functions will not increment.
- **Serial communications while in the function may be lost!**
- You should declare as volatile any variables that you modify within the attached function.
- By default, interrupts are atomic.

Atomic sections in Arduino

- We can enable/disable interrupts on certain sections of code with *interrupts()* and *noInterrupts()*.
- Interrupts in Arduino are enabled by default.
- Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.



Example

```
void setup() {}

void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Interrupts default

- By default you can **not** have interrupts inside interrupts.
- With C-Wiring both interrupt 0 and interrupt 1 are assigned the same priority.
- If you have a way to have interrupts inside interrupts, then when an interrupt is issued, you immediately leave the current interrupt and execute the new interrupt.
- Enabling interrupts inside interrupts is a "hack" : Is not recommended as it raises all sorts of issues with preserving the state of the machine before the interrupting interrupt is serviced.

Enabling interrupts inside interrupts

```
volatile int i,j,z;

void artificialdelay()
{
  for (i=0; i<900; i++){
    for (j=0; j<900; j++){
      z=i*10;}}
}

void setup()
{
  Serial.begin(9600);
  attachInterrupt(0, interrupt0, CHANGE);
  attachInterrupt(1, interrupt1, CHANGE);
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
}

void loop()
{
  Serial.println("entering main loop");
  artificialdelay();
  Serial.println("leaving main loop");
}
```

```
void interrupt0()
{
  interrupts();
    digitalWrite(13,HIGH);
    artificialdelay();
    digitalWrite(13,LOW);
  noInterrupts(); //not really needed
}

void interrupt1()
{
  interrupts();
    digitalWrite(12,HIGH);
    artificialdelay();
    digitalWrite(12,LOW);
   noInterrupts(); //not really needed
}
```

- If I press pin2 will go into `interrupt0()`.
- While it is processing, I can press pin3 and jump into function `interrupt1()`.

Operation scenarios

What happens if you trigger interrupt0 when inside interrupt0 ?

```
volatile int i,j,z;

void artificialdelay()
{
  for (i=0; i<900; i++){
    for (j=0; j<900; j++){
      z=i*10;}}
}

void setup()
{
  Serial.begin(9600);
  attachInterrupt(0, interrupt0, CHANGE);
  attachInterrupt(1, interrupt1, CHANGE);
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
}

void loop()
{
  Serial.println("entering main loop");
  artificialdelay();
  Serial.println("leaving main loop");
}
```

```
void interrupt0()
{
  interrupts();
    digitalWrite(13,HIGH);
    artificialdelay();
    digitalWrite(13,LOW);
  noInterrupts(); //not really needed
}

void interrupt1()
{
  interrupts();
    digitalWrite(12,HIGH);
    artificialdelay();
    digitalWrite(12,LOW);
   noInterrupts(); //not really needed
}
```

Nothing! Each interrupt has a register flag that indicates which interrupt needs attention. If we are inside a particular interrupt, the interrupt flag is already ON.

What happens if you trigger interrupt1 when inside interrupt0 ?

```
volatile int i,j,z;

void artificialdelay()
{
  for (i=0; i<900; i++){
    for (j=0; j<900; j++){
      z=i*10;}}
}

void setup()
{
  Serial.begin(9600);
  attachInterrupt(0, interrupt0, CHANGE);
  attachInterrupt(1, interrupt1, CHANGE);
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
}

void loop()
{
  Serial.println("entering main loop");
  artificialdelay();
  Serial.println("leaving main loop");
}
```

```
void interrupt0()
{
  //interrupts();
    digitalWrite(13,HIGH);
    artificialdelay();
    digitalWrite(13,LOW);
  //noInterrupts();
}

void interrupt1()
{
  //interrupts();
    digitalWrite(12,HIGH);
    artificialdelay();
    digitalWrite(12,LOW);
   //noInterrupts();
}
```

`interrupt1()` function will be executed immediately after we are done `interrupt0()`.