A practical introduction to embedded programming

Brian Plancher
Brian_Plancher@g.harvard.edu
10/17/2018
This week’s task is simple:
1. Since the boards you made 2 weeks ago are perfect and are still in perfect shape and are totally programmable...
2. And since you already know how to code in C...
3. Write some custom code to test a function on your board!... You did make sure that you can programmatically change the button and/or LED right (aka they are connected to PAx)?
This week’s task is simple:

1. Since the boards you made 2 weeks ago are perfect and are still in perfect shape and are totally programmable...
2. And since you already know how to code in C...
3. Write some custom code to test a function on your board!... You did make sure that you can programmatically change the button and/or LED right (aka they are connected to PAx)?

So as I said two weeks ago... if you are feeling like...
I HAVE NO IDEA

WHAT I'M DOING
RELAX
WE GOT THIS
One quick aside on boards before we talk about coding...

If you are goin to end up re-doing your board this is a really solid way to do it:
Now onto coding in AVR-C!

So if your first thought is: “What are codes”
Now onto coding in AVR-C!

So if your first thought is: “What are codes”

In short, **computer code is a human-readable language which tells the computer what to do**. The beauty of coding languages is that someone else wrote a **compiler** which translates the human readable words into 1s and 0s for the computer. The rules of a coding language are the assumptions the compiler makes during translation to ensure it gets it right!
Now onto coding in AVR-C!

So if your first thought is: “What is AVR-C? I feel like I should start with A…”

C is at this point the foundational language upon which most modern languages are based (or designed to be improvements on). AVR-C is a set of specific extensions to C to allow you to program your Attinys.
There are 5 basic datatypes you can use in C

- **int** - integer: a whole number.
- **float** - floating point value: i.e., a number with a fractional part.
- **double** - a double-precision floating point value.
- **char** - a single character.
- **void** - valueless special purpose type which we will examine closely in later sections.

**Data Types**
https://www.le.ac.uk/users/rjm1/cotter/page_19.htm

Remember for all things coding Google and Stackoverflow have MOST of the answers
You assign Variables (aka specific named instances of a type) to hold data

```
int my_age = 27;
char first_initial = 'B';
char last_initial = 'P';
```
You assign Variables (aka specific named instances of a type) to hold data

int my_age = 27;
char first_initial = 'B';
char last_initial = 'P';

Almost everything ends in semicolons in C!
Don't forget them!
You can then use conditional statements to make decisions about what to do with data.

Test expression is true

```java
int test = 5;
if (test < 10) {
    // codes
}
else {
    // codes
}
// codes after if...else
```

Test expression is false

```java
int test = 5;
if (test > 10) {
    // codes
}
else {
    // codes
}
// codes after if...else
```
You can then use conditional statements to make decisions about what to do with data

```java
int my_age = 27;
char first_initial = 'B';
char last_initial = 'P';
int above_drinking_age;
If (age > 21){
   above_drinking_age = 1;
} else {
   above_drinking_age = 0;
}
```
You can then use conditional statements to make decisions about what to do with data

```c
int my_age = 27;
char first_initial = 'B';
char last_initial = 'P';
int above_drinking_age;
if (age > 21) {
    above_drinking_age = 1;
} else {
    above_drinking_age = 0;
}
```

All if and else statements need the {} around them!
You can create functions to encapsulate some operate which you use a lot

```c
int checkID(int age){
    if (age > 21){
        return 1;
    } else {
        return 0;
    }
}
```

```c
int my_age = 27;
char first_initial = 'B';
char last_initial = 'P';
int above_drinking_age = checkID(my_age);
```
You can create functions to encapsulate some operate which you use a lot

```c
int checkID(int age){
    if (age < 21){
        return 1;
    } else {
        return 0;
    }
}
```

```c
int my_age = 27;
char first_initial = 'B';
char last_initial = 'P';
int above_drinking_age = checkID(my_age);
```

When you call a function you need to pass in the variables which it will use.
You can create functions to encapsulate some operate which you use a lot

```c
int checkID(int age){
    if (age < 21){
        return 1;
    } else {
        return 0;
    }
}

int my_age = 27;
char first_initial = 'B';
char last_initial = 'P';
int above_drinking_age = checkID(my_age);
```

You also need to specify the **return type** for the function and then make sure to return the appropriate thing.

When you **call** a function you need to pass in the variables which it will use.
Finally you use loops to repetitively call the same set of actions

```c
int class_ages[3];
```

This is an **ARRAY** which is a list of some type. In this case it is 3 ints.
Finally you use loops to repetitively call the same set of actions

```c
int class_ages[3];
class_ages[0] = 17;
class_ages[1] = 21;
class_ages[2] = 54;
```

This is an **ARRAY** which is a list of some type. In this case it is 3 ints.
It is zero-index!
Finally you use loops to repetitively call the same set of actions

```c
int class_ages[3];
class_ages[0] = 17;
class_ages[1] = 21;
class_ages[2] = 54;
int index = 0;
while (index < 3){
    if (checkID(class_ages[index])){
        letIntoBar();
    }
    index = index + 1;
}
```

We can use a **WHILE LOOP** to iterate until we hit the condition.
Finally you use loops to repetitively call the same set of actions

```c
int class_ages[3];
class_ages[0] = 17;
class_ages[1] = 21;
class_ages[2] = 54;
int index = 0;
while (index < 3){
    if (checkID(class_ages[index])){
        letIntoBar();
    }
    index++;
}
```
Finally you use loops to repetitively call the same set of actions

```c
int class_ages[3];
class_ages[0] = 17;
class_ages[1] = 21;
class_ages[2] = 54;
int index = 0;
while (index < 3){
    if (checkID(class_ages[index])){
        letIntoBar();
    }
    index++;
}
```

We can use a **WHILE LOOP** to iterate until we hit the condition

We can shorthand `index = index + 1;` to:
`index+=1;` or:
`Index++;`
Finally you use loops to repetitively call the same set of actions

```cpp
int class_ages[3];
class_ages[0] = 17;
class_ages[1] = 21;
class_ages[2] = 54;
for (int index = 0; index < 3; index++)
    if (checkID(class_ages[index])){
        letIntoBar();
    }
```
And that is programming in C in a nutshell
And that is programming in C in a nutshell

NOT BAD
//
// hello.ftdi.44.echo.c
//
// 115200 baud FTDI character echo, with flash string
//
// set lfuse to 0x5E for 20 MHz xtal
//
// Neil Gershenfeld
// 12/8/10
//
// (c) Massachusetts Institute of Technology
// This work may be reproduced, modified,
// performed, and displayed for any purpose
// retained and must be preserved. The
// as is; no warranty is provided, and
// liability.
//
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#define output(directions,pin) (directions |= pin) // set port direction for output
#define set(port,pin) (port |= pin) // set port pin
#define clear(port,pin) (port &= (~pin)) // clear port pin
#define pin_test(pins,pin) (pins & pin) // test for port pin
#define bit_test(byte,bit) (byte & (1 << bit)) // test for bit set
#define bit_delay_time 8.5 // bit delay for 115200 with overhead
#define bit_delay() _delay_us(bit_delay_time) // RS232 bit delay
#define half_bit_delay() _delay_us(bit_delay_time/2) // RS232 half bit delay
#define char_delay() _delay_ms(10) // char delay

Lets walk through Neil’s hello.ftdi.44.echo.c to explore AVR C code
//
// hello.ftdi.44.echo.c
//
// 115200 baud FTDI character echo, with flash string
//
// set lfuse to 0x5E for 20 MHz xtal
//
// Neil Gershenfeld
// 12/8/10
//
// (c) Massachusetts Institute of Technology 2010
// This work may be reproduced, modified, distributed,
// performed, and displayed for any purpose. Copyright is
// retained and must be preserved. The work is provided
// as is; no warranty is provided, and users accept all
// liability.
//
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#define output(directions,pin) (directions |= pin) // set port direction for output
#define set(port,pin) (port |= (-pin)) // set port pin
#define clear(port,pin) (port &= (-pin)) // clear port pin
#define pin_test(pins,pin) (pins & pin) // test for port pin
#define bit_test(byte,bit) (byte & (1 << bit)) // test for bit set
#define bit_delay_time 8.5 // bit delay for 115200 with overhead
#define bit_delay() _delay_us(bit_delay_time) // RS232 bit delay
#define half_bit_delay() _delay_us(bit_delay_time/2) // RS232 half bit delay
#define char_delay() _delay_ms(10) // char delay

// this is a single line comment
/*
This is a multi
line comment
*/

Comments are for YOU and for other people who will read your code later. Trust me you want to comment A LOT. It makes it much easier to debug. You will be happy later!

Note: as far as the program knows these don’t exist.
Includes are how you reuse code that someone else wrote.

We include .h files as they describe all the functions we need. Note: the actual code implementing those functions resides in a .c file.

As long as you are using only avr and util and other basic c programming stuff you won’t need to change your makefile. If you end up using random stuff from somewhere on the internet you will need to update your makefile to include that code.

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#define output(directions,pin) (directions |= pin) // set port direction
#define set(port,pin) (port |= pin) // set port pin
#define clear(port,pin) (port &= ~(-pin)) // clear port pin
#define pin_test(pins,pin) (pins & pin) // test for port pin
#define bit_test(byte,bit) (byte & (1 << bit)) // test for bit set
#define bit_delay_time 8.5 // bit delay for 115200 with overhead
#define bit_delay() _delay_us(bit_delay_time) // RS232 bit delay
#define half_bit_delay() _delay_us(bit_delay_time/2) // RS232 half bit delay
#define char_delay() _delay_ms(10) // char delay
```
Includes are how you reuse code that someone else wrote.

We include .h files as they describe all the functions we need. Note: the actual code implementing those functions resides in a .c file.

As long as you are using only avr and util and other basic c programming stuff you won’t need to change your makefile. If you end up using random stuff from somewhere on the internet you will need to update your makefile to include that code.

MAKE is one way to compile your code (remember the translation step to full computer 1s and 0s I talked about in the beginning)

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#define output(directions,pin) (directions |= pin) // set port direction
#define set(port,pin) (port |= pin) // set port pin
#define clear(port,pin) (port &=(~pin)) // clear port pin
#define pin_test(pins,pin) (pins & pin) // test for port pin
#define bit_test(byte,bit) (byte & (1 << bit)) // test for bit set
#define bit_delay_time 8.5 // bit delay for 115200 with overhead
#define bit_delay() _delay_us(bit_delay_time) // RS232 bit delay
#define half_bit_delay() _delay_us(bit_delay_time/2) // RS232 half bit delay
#define char_delay() _delay_ms(10) // char delay
```
Includes are how you reuse code that someone else wrote.

We include .h files as they describe all the functions we need. Note: the actual code implementing those functions resides in a .c file.

As long as you are using only avr and util and other basic c programming stuff you won’t need to change your makefile. If you end up using random stuff from somewhere on the internet you will need to update your makefile to include that code.

```
//
// hello.ftdi.44.echo.c
//
// 115200 baud FTDI character echo, with flash string
//
// set lfuse to 0x5E for 20 MHz xtal
//

#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#define output(directions,pin) (directions |= pin) // set port direction
#define set(port,pin) (port |= pin) // set port pin
#define clear(port,pin) (port &= ~(pin)) // clear port pin
#define pin_test(pins,pin) (pins & pin) // test for port pin
#define bit_test(byte,bit) (byte & (1 << bit)) // test for bit set
#define bit_delay_time 0.5 // bit delay for 115200 with overhead
#define bit_delay() _delay_us(bit_delay_time) // RS232 bit delay
#define half_bit_delay() _delay_us(bit_delay_time/2) // RS232 half bit delay
#define char_delay() _delay_ms(10) // char delay
```
Compiler does this for you automagically (by MAKE)! So all you have to do is write code that obeys the rules of C (and AVR)!

- C Code (.c, .h)
- Byte Code (.o)
- Hex Code (.hex)
Lets pause and take a look at the MAKEFILE (aka the instructions to MAKE) automagically (by MAKE)! So all you have to do is write code that obeys the rules of C (and AVR)!
PROJECT=hello.ftdi.44.echo
SOURCES= @(PROJECT).c
MMCU=attiny44
F_CPU = 20000000

CFLAGS=-mmcu=$(MMCU) -Wall -Os -DF_CPU=$(F_CPU)

@$ (PROJECT).hex: @$ (PROJECT).out
   avr-objcopy -O ihex @$ (PROJECT).out @$ (PROJECT).c.hex;
   avr-size --mcu=$(MMCU) --format=avr @$ (PROJECT).out

@$ (PROJECT).out: @$ (SOURCES)
   avr-gcc $ (CFLAGS) -I. -o @$ (PROJECT).out @$ (SOURCES)

program-usbtiny: @$ (PROJECT).hex
   avrdude -p t44 -P usb -c usbtiny -U flash:w: @$ (PROJECT).c.hex

program-usbtiny-fuses: @$ (PROJECT).hex
   avrdude -p t44 -P usb -c usbtiny -U lfuse:w:0x5E:m
PROJECT=hello.ftdi.44.echo
SOURCES=$(PROJECT).c
MMCUC=attiny44
F_CPU = 20000000

CFLAGS=-mmcu=$(MMCUC) -Wall -Os -DF_CPU=$(F_CPU)

$(PROJECT).hex: $(PROJECT).out
   avr-objcopy -O ihex $(PROJECT).out $(PROJECT).c.hex;
   avr-size --mcu=$(MMCUC) --format=avr $(PROJECT).out

$(PROJECT).out: $(SOURCES)
   avr-gcc $(CFLAGS) -I./ -o $(PROJECT).out $(SOURCES)

program-usbtiny: $(PROJECT).hex
   avrdude -p t44 -P usb -c usbtiny -U flash:w:$(PROJECT).c.hex

program-usbtiny-fuses: $(PROJECT).hex
   avrdude -p t44 -P usb -c usbtiny -U lfuse:w:0x5E:m
PROJECT=hello.ftdi.44.echo
SOURCES=$(PROJECT).c
MMCU=attiny44
F_CPU = 20000000

CFLAGS=-mmcu=$(MMCU) -Wall -Os -DF_CPU=$(F_CPU)

$(PROJECT).hex: $(PROJECT).out
    avr-objcopy -O ihex $(PROJECT).out $(PROJECT).c.hex;
    avr-size --mcu=$(MMCU) --format=avr $(PROJECT).out

$(PROJECT).out: $(SOURCES)
    avr-gcc $(CFLAGS) -I./ -o $(PROJECT).out $(SOURCES)

program-usbtiny: $(PROJECT).hex
    avrdude -p t44 -P usb -c usbtiny -U flash:w:$(PROJECT).c.hex

program-usbtiny-fuses: $(PROJECT).hex
    avrdude -p t44 -P usb -c usbtiny -U lfuse:w:0x5E:m
PROJECT=hello.ftdi.44.echo
SOURCES=$(PROJECT).c
MMCU=attiny44
F_CPU = 20000000

CFLAGS=-mmcu=$(MMCU) -Wall -Os -DF_CPU=$(F_CPU)

$(PROJECT).hex: $(PROJECT).out
    avr-objcopy -O ihex $(PROJECT).out $(PROJECT).c.hex;
    avr-size --mcu=$(MMCU) --format=avr $(PROJECT).out

$(PROJECT).out: $(SOURCES)
    avr-gcc $(CFLAGS) -I./ -o $(PROJECT).out $(SOURCES)

program-usbtiny: $(PROJECT).hex
    avrdude -p t44 -P usb -c usbtiny -U flash:w:$(PROJECT).c.hex

program-usbtiny-fuses: $(PROJECT).hex
    avrdude -p t44 -P usb -c usbtiny -U lfuse:w:0x5E:m

Tells the compiler to make a .o and a .hex file using avr (and automatically links in the standard c library things)
Takes a .hex file and sends it to the avr using with a program or fuse command

```bash
PROJECT=hello.ftdi.44.echo
SOURCES=$(PROJECT).c
MMCU=attiny44
F_CPU = 20000000

CFLAGS=-mmcu=$(MMCU) -Wall -Os -DF_CPU=$(F_CPU)

$(PROJECT).hex: $(PROJECT).out
  avr-objcopy -O ihex $(PROJECT).out $(PROJECT).c.hex;
  avr-size --mcu=$(MMCU) --format=avr $(PROJECT).out

$(PROJECT).out: $(SOURCES)
  avr-gcc $(CFLAGS) -I./ -o $(PROJECT).out $(SOURCES)

program-usbtiny: $(PROJECT).hex
  avrdude -p t44 -P usb -c usbtiny -U flash:w:$(PROJECT).c.hex

program-usbtiny-fuses: $(PROJECT).hex
  avrdude -p t44 -P usb -c usbtiny -U lfuse:w:0x5E:m
```
```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#define output(directions, pin) (directions |= pin) // set port direction for output
#define set(port, pin) (port |= pin) // set port pin
#define clear(port, pin) (port &= (~pin)) // clear port pin
#define pin_test(pins, pin) (pins & pin) // test for port pin
#define bit_test(byte, bit) (byte & (1 << bit)) // test for bit set
#define bit_delay_time 8.5 // bit delay for 115200 with overhead
#define bit_delay() _delay_us(bit_delay_time) // RS232 bit delay
#define half_bit_delay() _delay_us(bit_delay_time/2) // RS232 half bit delay
#define char_delay() _delay_ms(10) // char delay
```
#define is used to make some word a shorthand thing. Neil uses them here for a bunch of quick bitwise operations that we won’t have to worry about later. Think of them as super tiny functionors.

set(port,pin) will be replaced everywhere in the code with (port |= pin) but we can simply write the easier to remember set(port,pin)

Why is this helpful – lets talk binary numbers
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>Decimal</td>
<td>Binary</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
<tr>
<td>Decimal</td>
<td>Binary</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Symbol</th>
<th>Venn diagram</th>
<th>Boolean algebra</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td><img src="image" alt="AND Symbol" /></td>
<td><img src="image" alt="Venn Diagram" /></td>
<td>$A \cdot B$</td>
<td><img src="image" alt="Values" /></td>
</tr>
<tr>
<td>OR</td>
<td><img src="image" alt="OR Symbol" /></td>
<td><img src="image" alt="Venn Diagram" /></td>
<td>$A + B$</td>
<td><img src="image" alt="Values" /></td>
</tr>
<tr>
<td>XOR</td>
<td><img src="image" alt="XOR Symbol" /></td>
<td><img src="image" alt="Venn Diagram" /></td>
<td>$A \oplus B$</td>
<td><img src="image" alt="Values" /></td>
</tr>
<tr>
<td>NOT</td>
<td><img src="image" alt="NOT Symbol" /></td>
<td><img src="image" alt="Venn Diagram" /></td>
<td>$\overline{A}$</td>
<td><img src="image" alt="Values" /></td>
</tr>
<tr>
<td>Expression</td>
<td>Symbol</td>
<td>Venn diagram</td>
<td>Boolean algebra</td>
<td>Values</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------</td>
<td>----------------</td>
<td>--------</td>
</tr>
<tr>
<td>AND</td>
<td>![AND symbol]</td>
<td>![AND Venn diagram]</td>
<td>$A \cdot B$</td>
<td>&lt;table&gt;&lt;tr&gt;&lt;td&gt;A&lt;/td&gt;&lt;td&gt;B&lt;/td&gt;&lt;td&gt;Output&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;</td>
</tr>
<tr>
<td>OR</td>
<td>![OR symbol]</td>
<td>![OR Venn diagram]</td>
<td>$A + B$</td>
<td>&lt;table&gt;&lt;tr&gt;&lt;td&gt;A&lt;/td&gt;&lt;td&gt;B&lt;/td&gt;&lt;td&gt;Output&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;</td>
</tr>
<tr>
<td>XOR</td>
<td>![XOR symbol]</td>
<td>![XOR Venn diagram]</td>
<td>$A \oplus B$</td>
<td>&lt;table&gt;&lt;tr&gt;&lt;td&gt;A&lt;/td&gt;&lt;td&gt;B&lt;/td&gt;&lt;td&gt;Output&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;</td>
</tr>
<tr>
<td>NOT</td>
<td>![NOT symbol]</td>
<td>![NOT Venn diagram]</td>
<td>$\bar{A}$</td>
<td>&lt;table&gt;&lt;tr&gt;&lt;td&gt;A&lt;/td&gt;&lt;td&gt;Output&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;0&lt;/td&gt;&lt;td&gt;1&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;1&lt;/td&gt;&lt;td&gt;0&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;</td>
</tr>
</tbody>
</table>

```
define set(port, pin) (port |= pin) // set port pin
#define clear(port, pin) (port &= (~pin)) // clear port pin
```

| is logical OR |
| & is logical AND |
| ~ is logical NOT |

So if we pick a pin with a 1 then OR it we will set it. And if we AND the NOT of it we will AND a 0 and thus unset it!
### Boolean Algebra

<table>
<thead>
<tr>
<th>Expression</th>
<th>Symbol</th>
<th>Venn Diagram</th>
<th>Boolean Algebra</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td><img src="image" alt="AND Symbol" /></td>
<td><img src="image" alt="Venn Diagram for AND" /></td>
<td>$A \cdot B$</td>
<td>A</td>
</tr>
<tr>
<td>OR</td>
<td><img src="image" alt="OR Symbol" /></td>
<td><img src="image" alt="Venn Diagram for OR" /></td>
<td>$A + B$</td>
<td>A</td>
</tr>
<tr>
<td>XOR</td>
<td><img src="image" alt="XOR Symbol" /></td>
<td><img src="image" alt="Venn Diagram for XOR" /></td>
<td>$A \oplus B$</td>
<td>A</td>
</tr>
<tr>
<td>NOT</td>
<td><img src="image" alt="NOT Symbol" /></td>
<td><img src="image" alt="Venn Diagram for NOT" /></td>
<td>$\overline{A}$</td>
<td>A</td>
</tr>
</tbody>
</table>

- $|$ is logical OR
- $\&$ is logical AND
- $\sim$ is logical NOT

So if we pick a pin with a 1 then OR it we will set it. And if we AND the NOT of it we will AND a 0 and thus unset it!

But again Neil gives us this stuff so just remember to use it and you won’t have to worry about it as much! :-)

```c
#define set(port, pin) (port |= pin) // set port pin
#define clear(port, pin) (port &= (~pin)) // clear port pin
```
• Oh right this code was talking over serial with the computer and that was it so it only used two pins one for communication in (PA0) and one for communication out (PA1)

• Neil #defined them to words that he would remember up top so he didn’t have to keep thinking “wait was it PA0 or 1 for in” he could just use “serial_pin_in”

• But why is that format so weird? Well it turns out that AVR.h came with a bunch of shorthand so if you write it like that it works automatically. Otherwise you would have to consult the register table!

```
#define output(directions,pin) (directions |= pin) // set 
#define set(port,pin) (port |= pin) // set port pin 
#define clear(port,pin) (port &= (~pin)) // clear port 
#define pin_test(pins,pin) (pins & (1 << pin)) // test for port 
#define bit_test(byte,bit) (byte & (1 << bit)) // test 
#define bit_delay_time 8.5 // bit delay for 115200 with 
#define bit_delay() _delay_us(bit_delay_time) // RS232 
#define half_bit_delay() _delay_us(bit_delay_time/2) // 
#define char_delay() _delay_ms(10) // char delay 

#define serial_port PORTA 
#define serial_direction DDRA 
#define serial_pins PINA 
#define serial_pin_in (1 << PA0) 
#define serial_pin_out (1 << PA1) 
#define max_buffer 25
```
Remember from last time (electronics design) that the data sheet describes all of the ports and their names and what pins they are etc.
So now thanks to AVR.h we can just use the shorthand mapping!

Also the << is a bit shift but you don’t really have to worry about it for now and simply use it! :-)

(google bit masking if you are curious)
void get_char(volatile unsigned char *pins, unsigned char pin, char *rxbyte) {
   
   // read character into rxbyte on pins pin
   // assumes line driver (inverts bits)
   //
   LOTS OF STUFF WENT HERE

}

void put_char(volatile unsigned char *port, unsigned char txchar) {
   // send character in txchar on port pin
   // assumes line driver (inverts bits)
   //
   // start bit
   //
   LOTS OF STUFF WENT HERE
}

void put_string(volatile unsigned char *port, unsigned char *txstring) {
   // print a null-terminated string
   //
   LOTS OF STUFF WENT HERE
}

Neil did a bunch of stuff for you so if you use the baud rate 115200 (like from last week) this stuff just works and you don’t have to deal with synchronizing with the computer! Yay!

If you want at a later date we can talk about “bit-banging” but just know that this works and you can just use it to send characters. It even will work between two different Attinys.

Note: these are helper functions as they take in inputs and return outputs
The "main" function is what is actually run by the computer / Attiny. By standard it returns an integer. Also it has no inputs thus the "void" keyword is used.

Why is this last? —> C compiles top down
Variables that we will use in our function. Think of them as named things which we can assign values to in order to do things.

In the C language types MATTER. It will not compile without correct types.

```c
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;
    //
    // set clock divider to /1
    //
    CLKPR = (1 << CLKFCE);
    CLKPR = (0 << CLKPS3) | (0 << CLKPS2) | (0 << CLKPS1) | (0 << CLKPS0);
    //
    // initialize output pins
    //
    set(serial_port, serial_pin_out);
    output(serial_direction, serial_pin_out);
```
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;
    //
    // set clock divider to /1
    //
    CLKPR = (1 << CLKFCE);
    CLKPR = (0 << CLKPS3) | (0 << CLKPS2) | (0 << CLKPS1) | (0 << CLKPS0);
    //
    // initialize output pins
    //
    set(serial_port, serial_pin_out);
    output(serial_direction, serial_pin_out);

    "Hmmm this looks scary and I don’t think this program is doing anything crazy with timing or clocks so I’m just going to leave that as is."

    We can talk about it later
```c
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;
    //
    // set clock divider to /1
    //
    CLKPR = (1 << CLKFCE);
    CLKPR = (0 << CLKPS3) | (0 << CLKPS2) | (0 << CLKPS1) | (0 << CLKPS0);
    //
    // initialize output pins
    //
    set(serial_port, serial_pin_out);
    output(serial_direction, serial_pin_out);
}
```

Oh cool Neil used his shorthand #defines to make things make sense!

We are defining that the out pin is an output in both direction and port!
Oh cool Neil used his shorthand #defines to make things make sense!

We are defining that the out pin is an output in both direction and port!

For inputs it is a little more complicated depending on if you want pull-up resistors turned on
Remember from last time if your input is a GND for a signal you need the pullup resistor!

*cough* button *cough*
An example from my final project (I had a lot of buttons)

Also some fun short hand to reduce typing (you can | all of you setting because you want all of them to be a 1)

And you can set a conditional pound define (I had two Attiny’s on my button board)
In this case the computer sends us values so we don’t want the pullup on and so we do nothing (it is off by default)

But how do we tell what Ports / Pins we are using?

Well we defined it before by looking at the data sheet so we can just use our \#defined values and not worry about it!

```c
int main(void) {
   //
   // main
   //
   static char chr;
   static char buffer[max_buffer] = {0};
   static int index;
   //
   // set clock divider to /1
   //
   CLKPR = (1 << CLKFCE);
   CLKPR = (0 << CLKPS3) | (0 << CLKPS2) | (0 << CLKPS1) |
   //
   // initialize output pins
   //
   set(serial_port, serial_pin_out);
   output(serial_direction, serial_pin_out);
```
Once a variable is defined we can use it and assign it values

Note: again types matter!!!!!
While defines a LOOP (can also use “for”)

This is a core programming concept in C – we do things repetitively in loops and branch on conditional statements “if” and “else”

“While” will run until the condition in the “()” is FALSE so in this case it runs forever → thus our Attiny will repeat this action forever (one loop this small can run thousands of times a second so it better run for a long time or it will be too fast for us humans).

In general for AVR purposes we write all of the code that we want the AVR to do inside the while(1) loop.
int main(void) {
    //
    // main
    //
    static char chr;
    
    static char buffer[max_buffer] = {0};
    static int index;

    MORE STUFF WAS HERE

    // main loop
    //
    index = 0;

    while (1) {
        get_char(&serial_pins, serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.ftdi.44

        buffer[index++] = chr;

        if (index == (max_buffer-1))
            index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, \\
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
Buffer is an ARRAY (list) of char

```c
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;

    MORE STUFF WAS HERE

    //
    // main loop
    //
    index = 0;
    while (1) {
        get_char(&serial_pins, serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.ftdi.44.ec
        buffer[index++] = chr;
        if (index == (max_buffer-1))
            index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, \\
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
```

++ is shorthand for:
buffer[index] = chr;
index = index + 1;
Let’s use Neil’s helper function to get a value from the computer and save it in our chr variable

What about the &s

Pointer FUN?!
You don’t really need to know this just understand that the memory layout is complex and sometimes it is helpful to remember where you stored things and reference them indirectly.
var -> 50 // the variable itself has the value 50
ptr -> 1001 // the value of the ptr is the address of what it points to and therefore since it points to var it is 1001
&var -> 1001 // & operator gets us the address of that variable
*ptr -> 50 // * operator evaluates a pointer to get the value at this address
*(&var) -> 50 // The value at the address of var is just its value
Hmm this is a little complicated do I need to remember all of this right now?

var -> 50 // the variable itself has the value 50
ptr -> 1001 // the value of the ptr is the address of what it points to and therefore since it points to var it is 1001
&var -> 1001 // & operator gets us the adress of that variable
*ptr -> 50 // * operator evaluates a pointer to get the value at this address
*(&var) -> 50 // The value at the address of var is just its value
Not really just work off of the example code and copy the patterns but if you get confused later when you are doing some advanced code creation this slide is helpful!
void get_char(volatile unsigned char *pins, unsigned char pin, char *rxbyte) {

get_char(&serial_pins, serial_pin_in, &chr);

Looks like get_char wants a pointer variable type for the char it receives
char *pins means pointer to a char (as a type)
void get_char(volatile unsigned char *pins, unsigned char pin, char *rxbyte) {

get_char(&serial_pins, serial_pin_in, &chr);

So lets pass it the address of our local chr variable so it can save it there

Remember a pointer is really just an address!
So let's pass it the address of our local `chr` variable so it can save it there.

Remember a pointer is really just an address!

Ok but this still seems scary—oh wait we have Neil's example code and WE CAN JUST BASE OUR CODE ON HIS FOR NOW UNTIL WE FULLY UNDERSTAND IT!!!! :-)

```c
void get_char(volatile unsigned char *pins, unsigned char pin, char *rxbyte) {
  get_char(&serial_pins, serial_pin_in, &chr);
}
```
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;
    
    MORE STUFF WAS HERE
    
    //
    // main loop
    //
    index = 0;
    while (1) {
        get_char(&serial_pins, serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.ftdi.44.echo.");
        buffer[index++] = chr;
        if (index == (max_buffer-1))
            index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, '\"');
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;

    MORE STUFF WAS HERE

    //
    // main loop
    //
    index = 0;
    while (1) {
        get_char(&serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.ftdi.44.echo.c: you typed \");
        buffer[index++] = chr;
        if (index == (max_buffer-1))
            index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, \\
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;

    MORE STUFF WAS HERE

    //
    // main loop
    //
    index = 0;
    while (1) {
        get_char(&serial_pins, serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.ftdi.44.echo.c: you typed \"");
        buffer[index++] = chr;
        if (index == (max_buffer-1))
            index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, '\\n');
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
Neil is using this to say if you reach the end of the buffer go back to the beginning and loop around!

This means if the buffer was length 4 and we added the alphabet in we would get:

\[
[a,0,0,0] \rightarrow [a,b,0,0] \rightarrow [a,b,c,0] \rightarrow [a,b,c,d] \rightarrow [e,b,c,d] \rightarrow [e,f,c,d]
\]
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[4];
    static int i = 0;

    MORE STUFF WITH THE PORTS
    //
    // main loop
    //
    index = 0;
    while (1) {
        get_char(&serial_pins, serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.ftdi.4
        buffer[index++1] = chr;
        \textcolor{red}{\textbf{if}} (index == (\text{max} _\text{buffer}-1))
        \quad index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, '\n\n');
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
```c
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;

    MORE STUFF WAS HERE

    //
    // main loop
    //
    index = 0;
    while (1) {
        get_char(&serial_pins, serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.ftdi.44.echo.c: you typed \"");
        buffer[index++] = chr;
        if (index == (max_buffer-1))
            index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, '\n');
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
```
int main(void) {
    //
    // main
    //
    static char chr;
    static char buffer[max_buffer] = {0};
    static int index;

    MORE STUFF WAS HERE

    //
    // main loop
    //
    index = 0;
    while (1) {
        get_char(&serial_pins, serial_pin_in, &chr);
        put_string(&serial_port, serial_pin_out, "hello.");
        buffer[index++] = chr;
        if (index == (max_buffer-1))
            index = 0;
        put_string(&serial_port, serial_pin_out, buffer);
        put_char(&serial_port, serial_pin_out, "\n");
        put_char(&serial_port, serial_pin_out, 10); // new line
    }
}
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>Chr</th>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>HTML</th>
<th>Chr</th>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>HTML</th>
<th>Chr</th>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>HTML</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>000</td>
<td>NULL</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>&amp;#032;</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>&amp;#064;</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>140</td>
<td>&amp;#096;</td>
<td>`,</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>Start of Header</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>&amp;#033;</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>&amp;#065;</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>141</td>
<td>&amp;#097;</td>
<td>`a</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>002</td>
<td>Start of Text</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>&amp;#034;</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>&amp;#066;</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>142</td>
<td>&amp;#098;</td>
<td>`b</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>003</td>
<td>End of Text</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>&amp;#035;</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>&amp;#067;</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>143</td>
<td>&amp;#099;</td>
<td>`c</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>004</td>
<td>End of Transmission</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>&amp;#036;</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>&amp;#068;</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>144</td>
<td>&amp;#100;</td>
<td>`d</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>005</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>&amp;#037;</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>&amp;#069;</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>145</td>
<td>&amp;#101;</td>
<td>`e</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>006</td>
<td>Acknowledgment</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>&amp;#038;</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>&amp;#070;</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>146</td>
<td>&amp;#102;</td>
<td>`f</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>007</td>
<td>Bell</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>&amp;#039;</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>&amp;#071;</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>147</td>
<td>&amp;#103;</td>
<td>`g</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>010</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>&amp;#040;</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>&amp;#072;</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>150</td>
<td>&amp;#104;</td>
<td>`h</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>011</td>
<td>Horizontal Tab</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>&amp;#073;</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>151</td>
<td>&amp;#105;</td>
<td>i</td>
<td>106</td>
<td>6A</td>
<td>152</td>
<td>&amp;#106;</td>
<td>`j</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>012</td>
<td>Line feed</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>&amp;#074;</td>
<td>J</td>
<td>107</td>
<td>6B</td>
<td>153</td>
<td>&amp;#107;</td>
<td>k</td>
<td>108</td>
<td>6C</td>
<td>154</td>
<td>&amp;#108;</td>
<td>`l</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>013</td>
<td>Vertical Tab</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>&amp;#075;</td>
<td>K</td>
<td>109</td>
<td>6D</td>
<td>155</td>
<td>&amp;#109;</td>
<td>m</td>
<td>110</td>
<td>6E</td>
<td>156</td>
<td>&amp;#110;</td>
<td>`n</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>014</td>
<td>Form feed</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>&amp;#076;</td>
<td>L</td>
<td>111</td>
<td>6F</td>
<td>157</td>
<td>&amp;#111;</td>
<td>o</td>
<td>112</td>
<td>70</td>
<td>160</td>
<td>&amp;#112;</td>
<td>`p</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>015</td>
<td>Carriage return</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>&amp;#077;</td>
<td>M</td>
<td>113</td>
<td>71</td>
<td>161</td>
<td>&amp;#113;</td>
<td>q</td>
<td>114</td>
<td>72</td>
<td>162</td>
<td>&amp;#114;</td>
<td>`r</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>016</td>
<td>Shift Out</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>&amp;#046;</td>
<td>.</td>
<td>115</td>
<td>73</td>
<td>163</td>
<td>&amp;#115;</td>
<td>s</td>
<td>116</td>
<td>74</td>
<td>164</td>
<td>&amp;#116;</td>
<td>`t</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>017</td>
<td>Shift In</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>&amp;#047;</td>
<td>/</td>
<td>117</td>
<td>75</td>
<td>165</td>
<td>&amp;#117;</td>
<td>u</td>
<td>118</td>
<td>76</td>
<td>166</td>
<td>&amp;#118;</td>
<td>`v</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>Data Link Escape</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>&amp;#048;</td>
<td>0</td>
<td>119</td>
<td>77</td>
<td>167</td>
<td>&amp;#119;</td>
<td>w</td>
<td>120</td>
<td>78</td>
<td>170</td>
<td>&amp;#120;</td>
<td>`x</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>Device Control 1</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>&amp;#049;</td>
<td>1</td>
<td>121</td>
<td>79</td>
<td>171</td>
<td>&amp;#121;</td>
<td>y</td>
<td>122</td>
<td>7A</td>
<td>172</td>
<td>&amp;#122;</td>
<td>`z</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>Device Control 2</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>&amp;#050;</td>
<td>2</td>
<td>123</td>
<td>7B</td>
<td>173</td>
<td>&amp;#123;</td>
<td>{</td>
<td>124</td>
<td>7C</td>
<td>174</td>
<td>&amp;#124;</td>
<td>`</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>Device Control 3</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>&amp;#051;</td>
<td>3</td>
<td>125</td>
<td>7D</td>
<td>175</td>
<td>&amp;#125;</td>
<td>}</td>
<td>126</td>
<td>7E</td>
<td>176</td>
<td>&amp;#126;</td>
<td>~</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>Device Control 4</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>&amp;#052;</td>
<td>4</td>
<td>127</td>
<td>7F</td>
<td>177</td>
<td>&amp;#127;</td>
<td>Del</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: asciiichars.com
Key things to make sure you are doing in your code!!

• USE BRACKETS {}

• USE SEMICOLONS ;

• All helper things come before Main

• GOOGLE IS YOUR FRIEND!
So what else is in that data sheet?
Timers and Clock Registers

11.9.3 TCNT0 – Timer/Counter Register

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the Compare Match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a Compare Match between TCNT0 and the OCR0x Registers.

11.9.4 OCR0A – Output Compare Register A

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.
<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Program Address</th>
<th>Label</th>
<th>Interrupt Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0000</td>
<td>RESET</td>
<td>External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset</td>
</tr>
<tr>
<td>2</td>
<td>0x0001</td>
<td>INTO</td>
<td>External Interrupt Request 0</td>
</tr>
<tr>
<td>3</td>
<td>0x0002</td>
<td>PCINT0</td>
<td>Pin Change Interrupt Request 0</td>
</tr>
<tr>
<td>4</td>
<td>0x0003</td>
<td>PCINT1</td>
<td>Pin Change Interrupt Request 1</td>
</tr>
<tr>
<td>5</td>
<td>0x0004</td>
<td>WDT</td>
<td>Watchdog Time-out</td>
</tr>
<tr>
<td>6</td>
<td>0x0005</td>
<td>TIM1_CAPT</td>
<td>Timer/Counter1 Capture Event</td>
</tr>
<tr>
<td>7</td>
<td>0x0006</td>
<td>TIM1_COMPA</td>
<td>Timer/Counter1 Compare Match A</td>
</tr>
<tr>
<td>8</td>
<td>0x0007</td>
<td>TIM1_COMPB</td>
<td>Timer/Counter1 Compare Match B</td>
</tr>
<tr>
<td>9</td>
<td>0x0008</td>
<td>TIM1_OVF</td>
<td>Timer/Counter1 Overflow</td>
</tr>
<tr>
<td>10</td>
<td>0x0009</td>
<td>TIM0_COMPA</td>
<td>Timer/Counter0 Compare Match A</td>
</tr>
<tr>
<td>11</td>
<td>0x000A</td>
<td>TIM0_COMPB</td>
<td>Timer/Counter0 Compare Match B</td>
</tr>
<tr>
<td>12</td>
<td>0x000B</td>
<td>TIM0_OVF</td>
<td>Timer/Counter0 Overflow</td>
</tr>
<tr>
<td>13</td>
<td>0x000C</td>
<td>ANA_COMP</td>
<td>Analog Comparator</td>
</tr>
<tr>
<td>14</td>
<td>0x000D</td>
<td>ADC</td>
<td>ADC Conversion Complete</td>
</tr>
<tr>
<td>15</td>
<td>0x000E</td>
<td>EE_RDY</td>
<td>EEPROM Ready</td>
</tr>
<tr>
<td>16</td>
<td>0x000F</td>
<td>USI_STR</td>
<td>USI START</td>
</tr>
<tr>
<td>17</td>
<td>0x0010</td>
<td>USI_OVF</td>
<td>USI Overflow</td>
</tr>
</tbody>
</table>
And so so so much more (e.g. ADC) so read up!
:-)
Embedded Programming

AVR Programming: Learning to Write Software for Hardware 1st Edition
by Elliot Williams (Author)
Possible Lightweight Editors to Use (IDE)

Everything is harder on windows → Linux VM
And we’re done!

Questions?