C notes

Book 1

Notes from text

Ternary operator

* And if-else statement (statement) ? true : false

Literal

* A constant value assigned to a variable

Macros

* A #define function

Array def

* Arrays use pointer in the form of [… , char const\*const, …]
* For the array bird = {raven,crow} also looks like

bird = {char const\*const, char const\*const}

Array Length

* Sizeof A / sizeof A[0]; size of array divided by the size of the objects

Pointer review(I know)

* Pointers to arrays, point to the first element
* Incrementing (p++) a pointer for an array value goes to the next value
* p[0]=\*a and p[1]=\*(a+1), this will be a different number of bytes depending on how bit the object types are, char array will grow by one, but int arrays will grow 4
* when using a pointer to a structure, instead of person1.name, you do

person1->name. this saves a lot of space.

* void pointers can point to any type of pointer (ex, const void\* x)
* Function pointer is initializedby [ type (\*p name)(param,prarm…) = address pt ]
  + The pointer works the same as the function but you can use it to pass functions to other functions
  + [ Int sum = add(5 ,3) ] works the same as [ int sum = (\*pAdd)(5 ,3) ]
  + You can now create a function such as
    - Int add( char x, const void\* y)

Functions and operators

Size\_t

* A type of data that represents the size (strlen(x) returns size\_t).
* For sizes, cardinals, and ordinal numbers

Qualifier

* Const is used for variables that we don’t want to change ever

Typedef

* Lets use a new name for an existing data type
* Instead of unsinged long int = 4; we can write typedef (unsigned long int) big\_t; letting us use big\_t as a new data type.
* It can also be used to not have to use struct before a type when using a structure.

Void\*

* Pointer to an unknown type

Book 2

Notes from text

hello

Functions and operators

Extern

* (ex. Extern char x) tells the compiler that the variable is defined later or in another file.

Read later

5.7 binary representation

12 the c memory model

Everything after chapter 12

Book 2

Chapter 5

He said 5,6,9,10

Study up on

* How do header files work
* Using functions between files
* Read electrical circuitry

Questions to ask about the code

1. The functions about input and output bits are used often from the io.c file, what do they do and how do they work?
   1. It changed the mask not the input.
   2. How is that mask later used in the code /hardware in??
   3. is each input given a different binary value (000100), and then the system toggles that value to set in the inputPinMask when engaged?
   4. What does the limit input mean?
   5. Does this relate to the output pin?
2. What do we sense from the controller and what do we output? 5 inputs and three outputs?
3. I understand that the modbus used to transmit binary data between devices, but how is it used in the code?

I’d like to know more about

Firmware

* HoldingRegister / inputRegister
* ModBusHandeler
* Outputs
* FlashStack
* IO
* Safety.c and safety.h / SAFE\_CALL()
* StateMachine
* SystemConfig

Random stuff

* Parts to the compiler

1. Lexar
   * Takes the files and converts each line into tokens and tells what lines they are on
   * Contains name, what the token is, then the location
   * Ex. “numeric\_contant ‘0’ Loc=<./code.cpp: 3:16>”
2. Parser
   * Conver the tokens into AST, a physical tree that verifies that the tokens match the language grammar
   * Output error if incorrect
3. Code gen
   * Machine code which takes the ASL tree and transforms it into code for the computer to run
4. Linker
   * Makes each c file separately into object files
   * Linkes the object files together to produce a single executable file(.exe)

Leet code

Codex

stm32 cortex-M

* addresses, are set to volatile to stop the pre-processor from editing them.
* SysTick timer, are timers that run in the background so that you can set them at the start and then use if loops to find when they end instead of sleeping the code
* Interrupts, stop the code when the SysTick timer is done.
* PWM, pulse width modulation
  + Period – peak to peak time
  + Duty cycle- how long is it high vs low long is it low in %
  + To change how bright a light is you can change the duty cycle so instead of the light only being 50% of the time it’s on 80% of the time
  + Multiple timers can be used to create different PWM outputs
* Timers, different timer have different ports for import and export
* Bootloader,
  + Place to start up the code and update the firmware
  + Given a separate storage space in the flash memory
  + Later code removed the bootloader from the bottom of the stack
  + Make a function pointer to the firmware main loop
  + In assembly .s file .incbin “file” and .section .bootloader makes a bootloader object file to then link from the link script file that tells the linker what to do first
  + Padding is needed to fill the black space after the bootloader and before the firmware.
* UART driver, send data bits in serial without a timer by having a standard timer set for the read device which reads every half bit period. Stops the code for the duration of the send, can miss the signal. Needs a buffer to hold info when not searching for received information.
* Ring buffer,
* Car with Bluetooth remote