Chapter 1

Embedded software

1.1 C review

1.1.1 Declaring variables

type-qualifier(s) type-modifier data-type variable-name = initial-value;
Modifiers:

- short - 2 byte
- long - 4 byte
- unsigned
- signed

Qualifiers:

- const
- volatile
- restrict

Data types:

- char - 1 byte
- int - 2/4 byte
-

```
    const unsigned char foo = 12;
    long int foo;
    ...
    foo = 400;
```

1.1.2 Operators

Type	Operators
Logical	||, &&, !
Bitwise	«, », |, &, ^
Arithmetic	+,-,/,*,++,–,%

1.1.3 Functions and headers

```c
#include "file.h"

/* function definition*/
void foo (int *a, char b) {
    *a = b % 2;
}
```

Listing 1.1: file.c

```c
#ifndef __FILE_H__
#define __FILE_H__

/* function definition*/
void foo (int *a, char b);
#endif /* __FILE_H__ */
```

Listing 1.2: file.h

1.1.4 pointers

```c
int foo = 0x34;
int * ptr;     \\pointer declartion
ptr = &foo;    \\address of the operator
*ptr = 0x52;    \\ dereference operator
```

now the value of foo=0x52

1.1.5 example c code maintainability

```c
/**************************************************************
* File: exmple.c
* Copyright 2025 Thakshana technologies
* All Rights Reserved.
*
* The informnation in this file meant as a exaple c ode used according to the thakshana's architecture. copying and distributing of this file withouy the concent of the Thakshana technologies is prohibited.
*
* Author: Vakeesan Karunanithy
* Date: Edited January 2025
*
* Description: A simple code section for the upcoming coding practice in my career for c.
*               basic features:
*                   -Average
*                   - Maximum
*               Note: Fill it if you need it
******************************************************************/
/*****************************************************************
* Function: find average
* Description:
*           This fucntion taskes a set of numbers and performs finding the average of the set
* Parameters:
*       int *ptr: pointer to a dataset
*       int count: number of item in the dataset
* Return:
```

```
*       Average of the numbers provided.
*********************************************************************/
int find_average (int * ptr, int count);
/********************************************************************
*
* Constatnts
*
*********************************************************************/
#define NULL (0)
#define NUMBER_SET_LENGTH (20)

void main(){
    /*array of numbers*/
    int numbers[NUMBER_SET_LENGTH] = {};




}
```

Listing 1.3: example.c

1.2 GCC and GNU Make

build process

- preprocessor (*.c/*.h -> *.i)
- compiler (*.i -> *.s)
- assembler (*.s -> *.o)
- linker (*.o & *.a -> relocatable file)
- locator
- installing

1.3 Compiling and GCC

1.3.1 GCC tool check

<ARCH>-<VENDOR>-<OS>-<ABI>
ex: arm-none-eabi-gcc
none- baremetal

General compiler flags

Format	Purpose
-c	compile and assemble not link
-o <FILE>	compile, assemble, and link to output file
-g	general debugging information
-Wall	enable all warning messages
-Werror	treat all warnings as errors
-I <DIR>	include this dir to look for header files
-std=STANDARD	which standard to use
-v	verbose output

```
    gcc -std=c99 -Werror -o main.out main.c
```

Architecture specific compiler flags

Format	Purpose
-mcpu	cortex-a8
-march	armv8, thumb
-mtune	cortex-m0plus
-mthumb	thumb state
-marm	arm state
-mlittle-endian	little endian
-mbig-endian	big endian

1.4 Preprocessor directives

#
stop after preprocessing

```
    gcc -E -o main.i main.c
```

define as a constant

```
    #define LENGTH (10)
    /*macro defined as another macro*/
    #define UART_ERROR ERROR
```

define as macro function

```
    #define SQUARE(x) (x*x)
    ....
    y = SQUARE(2);
```

define as boolean compilation conditions

```
    /* define feature for msp*/
    #define MSP_PLATFORM

    /* undefine constant*/
    #undef KL25_PLATFORM
```

conditionally compile blocks

```
    #ifdef
    #ifndef
    #elif
    #else
    #endif
```

compile time switch

```
    gcc -DMSP_PLATFORM -o main.out main.c
```

1.5 creating header

```
    #pragma once
    char memzero(char * src);
```

Listing 1.4: memory.h

1.6 Linker

details about how to map compiles data into the physical memory. here is the part of linker script.

```
    MEMORY
    {
        MAIN (RX):origin=0x00000000, length=0x0040000
        SRAM_DATA (RW):origin=0x2000000,length=0x0001000
    }
```

Listing 1.5: physical memory regions

there is compiled memory section as well.

```
    SECTIONS
    {
        .intvecs: > 0x00000000
        .text : > MAIN
        .const : > MAIN
        .data : >SRAM_DATA
        .bss : >SRAM_DATA
        .heap : >SRAM_DATA
        .stack : >SRAM_DATA(HIGH)

    }
```

1.6.1 linker flags

format	purpose
-map [NAME]	outputs the memory map file from the result of linking
-T [NAME]	specifies a linker script name [NAME
-o [NAME]	place the output on the file name
-O<>	level of optimization [=0-3]
-Os	optimize for memory size
-z stacksize=[SIZE]	amount of stack space to reserve
-shared	produce shared library
-l [LIB]	link with library
-L [DIR]	inlcude the following lib path
-Wl,<OPTION>	pass the option to linker from compiler
-Xlinker <OPTION>	pass option to linker from compiler

1.7 Make

building is tedious makefile, is no target is pecified then execute the first defined target in the makefile to be executed.

```
    make main.o
    make all
    make clean
    make
```

here the targets can have dependecies which are like below.

```
    main.out: main.o my_file.o
        gcc -g -Wl -o main.out main.o my_file.o
```

build rule specify the specific syntax of target: prerequisite and commands. these commands known as recipes.

1.7.1 Makefile syntax

```
#this is comment

#includes another file
include sources.mk

#variable and line continuation
FLAGS = -g \
        -Werror \
        -std=c99
#my_file.o target binary
my_file.o: myfile.h myfile.c
    gcc $(FLAGS) -c -o myfile.o myfile.c

#mian.o target
...
```

Makefile variables
= for recursively expanded variables := for the simply expanded variables - I guess which means the shell commands
ARCH:=$(shell arch) then compiler and linker flags. CFLAGS = -g -std=$(CSTD)-mcpu$ =(CPU) -mthumb
Include paths and Sources

```
INCLUDES=   \
    -I ./libs   \
    -I ./modem  \
    -I ./uart   \
    -I ./arch
SRCS=       \
./main.c    \
./memory.c  \
./uart.c    \
./data.c
```

use variables in target rules

```
$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) $(INCLUDES) $(LDFLAGS) -o   $(TARGET) $(OBJS)
```

$@ - Target
$^ All prerequisites
Pattern matching operator is %. target object rule with an associated source file.

```
%.o: %.c
    $(CC) -C  $@ -o $< $(CFLAGS)
```

here when i call the make it will automatically use the name of the traget and sources from the input command. so the above line of codes will change to as below.

```
main.o: main.c
    $(CC) -C main.c -o main.o $(CFLAGS)
```

another useful in the pattern matching, we can use the source variables to generate a list of object files variable.

```
OBJS:=$(SRCS:.c=.o)
```

target do not have to be a file. but we have to put the .PHONY directive.

```
    .PHONY: all
    all: main.out
    main.out: $(OBJS)
        gcc $(CFLAGS) -o main.out $(OBJS)
```

for more understanding see the next code section.

```
    .PHONY: clean

clean:
    rm -f *.o
    rm -f my_program
```

In this example, clean is a "phony" target. Even if there's a file named clean in the directory, Make will always run the rm commands associated with the clean target.(chatgpt text) another interesting thing is functions and dynamic variables.

```
    OS:=$(shell uname -s)
    ifeq ($(OS),Linux)
        CC=gcc
    endif
```

overriding variables

```
    make all PLATFORM=msp432

    # input can set variables
    ifeq ($(PLATFORM),MSP)
        CPU=cortex-m4
    endif

    ifeq($(PLATFORM),FRDM)
        CPU=cortex-m0plus
    endif
```

finally the number of files

```
    cat sources.mk

    SRCS= main.c \
            myfile.c \
            my_memory.c
```

example makefile

```
#--------------------------------------------
# Simple makefile for build system
#
# Use: make [targets] [overrides]
#
# Targets:
#   <FILE>.o blah blah
#
# Overrides:
#   CPU - ARM cortex architecture
#
#---------------------------------------------
include sources.mk

#overrides
CPU = cortex-m0plus
ARCH = thumb
SPECS = nosys.specs

```

```
#compile defines
CC = arm-none-eabi-gcc
LD = arm-none-eabi-ld
BASENAME = demo
TARGET= $(BASENAME).out
LDFLAGS = -Wl, -Map=$(BASENAME).map
CFLAGS = -mcpu=$(CPU) -m$(ARCH) --specs=$(SPECS) -Wall

OBJS = $(SRCS:.c=.o)

%.o: %.c
    $(CC) -c $< $(CFLAGS) -o $@
.PHONY:build
build:all
.PHONY:all
all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) $(OBJS) $(CFLAGS) $(LDFLAGS) -o $@
.PHONY:clean
clean:
    rm -f $(OBJS) $(TARGET) $(BASENAME).map
```