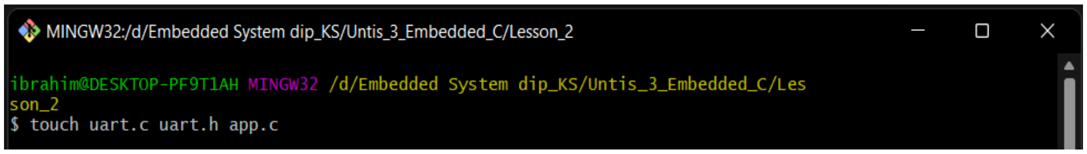
Lab_1

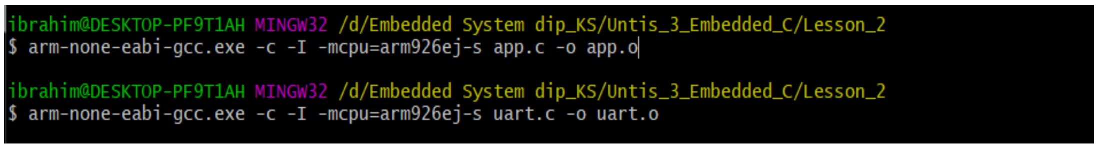
➢ In this lab I want to make simple project to print learn-in-depth: Ibrahim Mohamed without using IDE, So I create startup code, linker script and files of project that consist of app.c, uart.c and uart.h.

1. Create the 3 files by terminal.



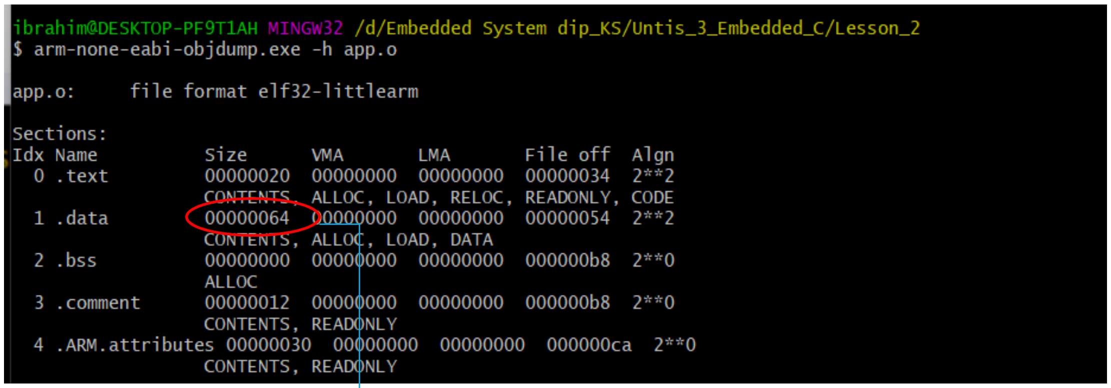
```
MINGW32:/d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2

ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Les
son_2
$ touch uart.c uart.h app.c
```

2. After writing the code, I compiled them and got object file.



```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-gcc.exe -c -I -mcpu=arm926ej-s app.c -o app.o

ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-gcc.exe -c -I -mcpu=arm926ej-s uart.c -o uart.o
```

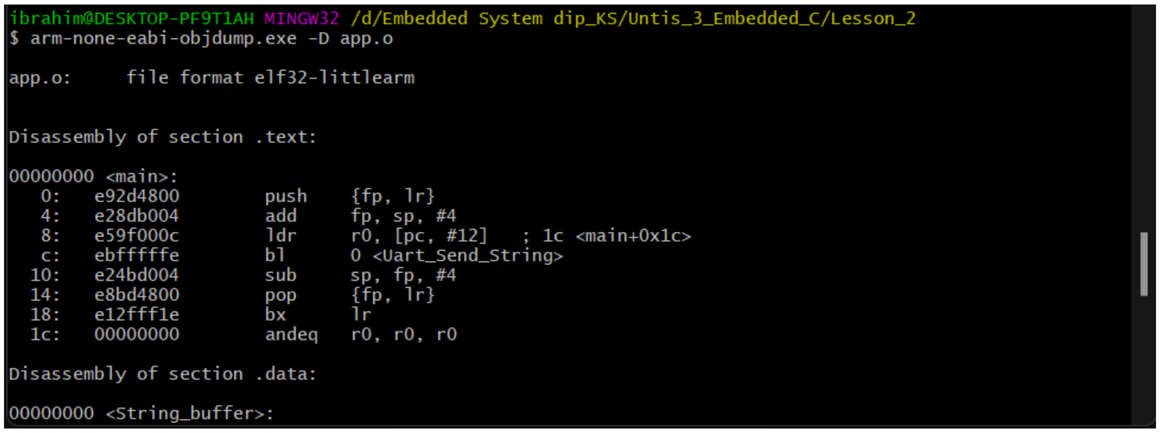
3. These sections on app.o.



```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000020  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000064  00000000  00000000  00000054  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b8  2**0
                  ALLOC
  3 .comment      00000012  00000000  00000000  000000b8  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000030  00000000  00000000  000000ca  2**0
                  CONTENTS, READONLY
```

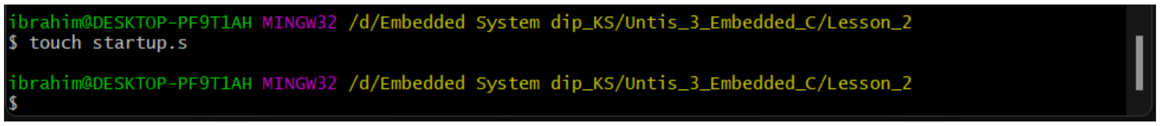
Note: in data section I have only 64 bytes in hex that equal 100 in dec. that right because I initialized global variable its size = 100 bytes.

4. Hear I show you disassembly in app.o.



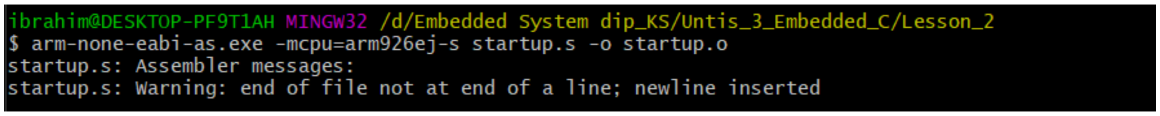
```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-objdump.exe -D app.o

app.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0:   e92d4800        push    {fp, lr}
   4:   e28db004        add     fp, sp, #4
   8:   e59f000c        ldr     r0, [pc, #12]   ; 1c <main+0x1c>
   c:   ebfffffe        bl      0 <Uart_Send_String>
  10:   e24bd004        sub     sp, fp, #4
  14:   e8bd4800        pop     {fp, lr}
  18:   e12fff1e        bx      lr
  1c:   00000000        andeq   r0, r0, r0

Disassembly of section .data:

00000000 <String_buffer>:
```

5. I create startup file.



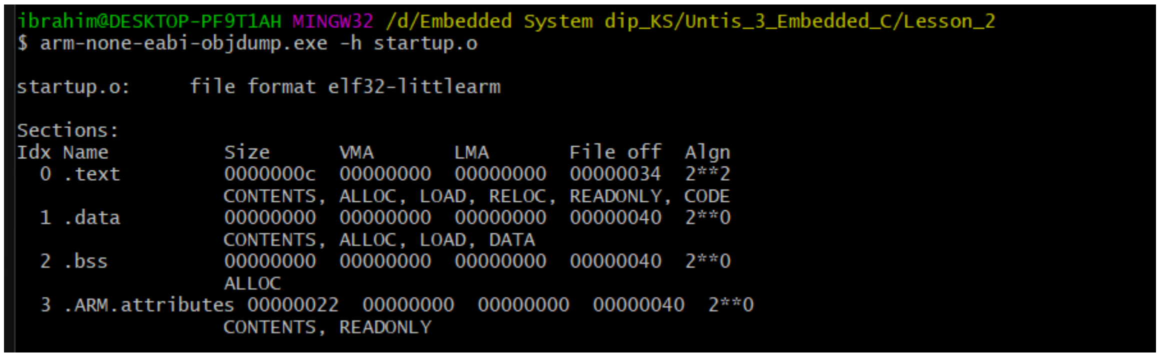
```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ touch startup.s

ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$
```

6. After writing startup code, I compiled it and get startup.o.



```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted
```

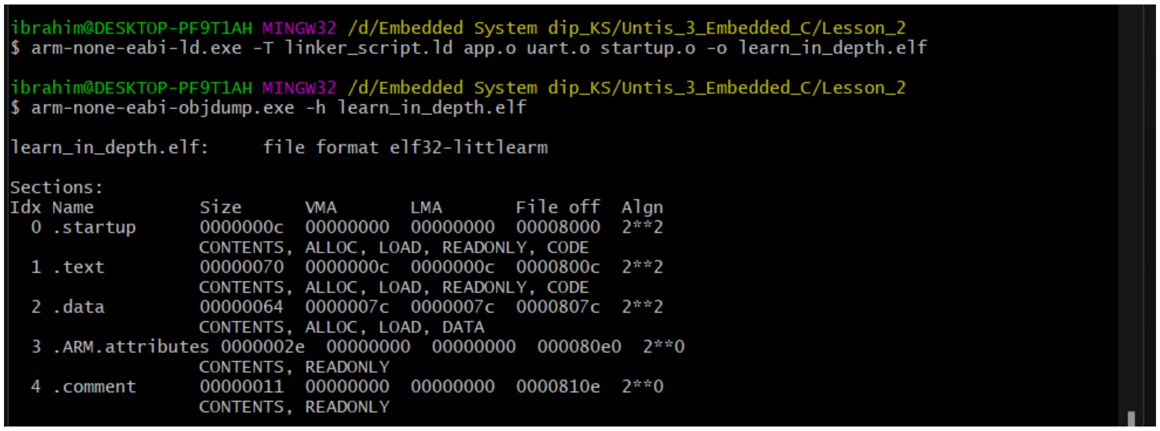
7. These sections of startup code.



```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-objdump.exe -h startup.o

startup.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000000c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000040  2**0
                  ALLOC
  3 .ARM.attributes 00000022  00000000  00000000  00000040  2**0
                  CONTENTS, READONLY
```

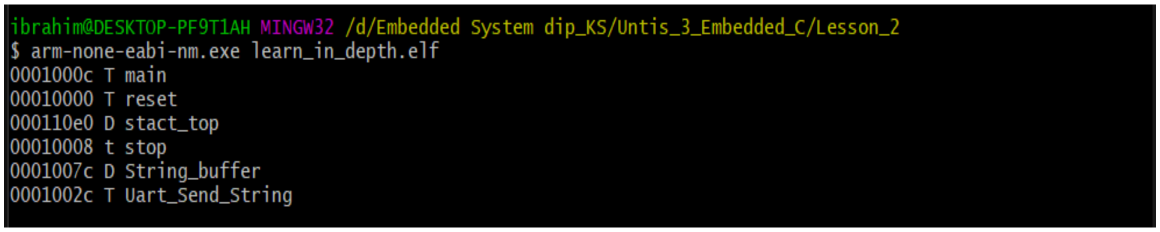
8. Then I create the final file is linker script to link all files each other and make them in one file.

9. Hear is the step of linked files and sections of output



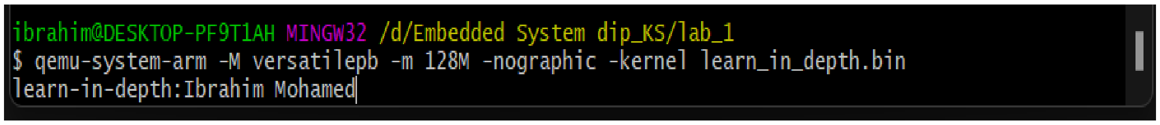
```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-ld.exe -T linker_script.ld app.o uart.o startup.o -o learn_in_depth.elf

ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-objdump.exe -h learn_in_depth.elf

learn_in_depth.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .startup      0000000c  00000000  00000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         00000070  0000000c  0000000c  0000800c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         00000064  0000007c  0000007c  0000807c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  000080e0  2**0
                  CONTENTS, READONLY
  4 .comment      00000011  00000000  00000000  0000810e  2**0
                  CONTENTS, READONLY
```

10. Hear I show you the symbols in output file.



```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/Untis_3_Embedded_C/Lesson_2
$ arm-none-eabi-nm.exe learn_in_depth.elf
0001000c T main
00010000 T reset
000110e0 D stact_top
00010008 t stop
0001007c D String_buffer
0001002c T Uart_Send_String
```

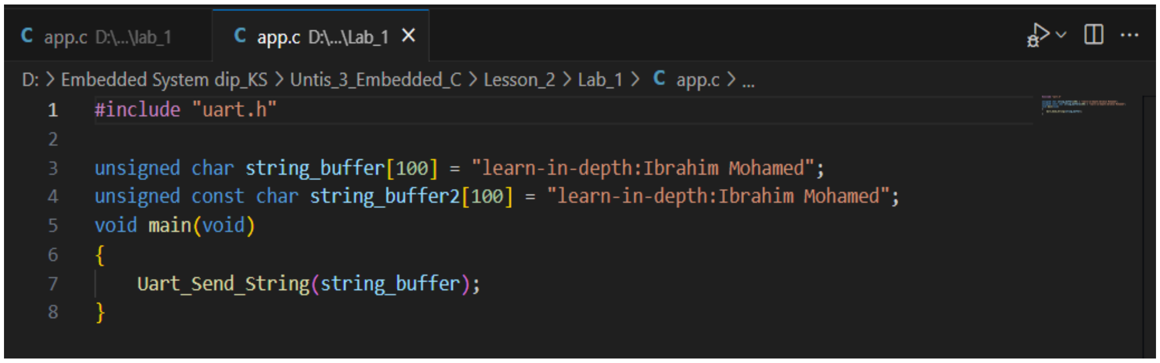
11. Finally, I burn this project on Qemu and show the output of this project.



```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Embedded System dip_KS/lab_1
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn_in_depth.bin
learn-in-depth:Ibrahim Mohamed
```

Codes

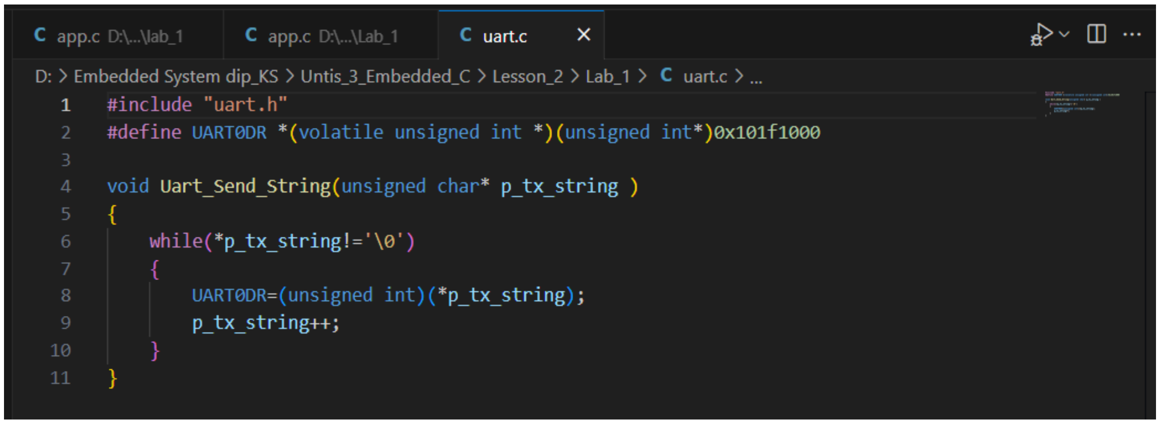
➢ app.c



app.c D:\...\lab_1
app.c D:\...\Lab_1
D: > Embedded System dip_KS > Untis_3_Embedded_C > Lesson_2 > Lab_1 > app.c > ...

```c
#include "uart.h"

unsigned char string_buffer[100] = "learn-in-depth:Ibrahim Mohamed";
unsigned const char string_buffer2[100] = "learn-in-depth:Ibrahim Mohamed";
void main(void)
{
    Uart_Send_String(string_buffer);
}
```

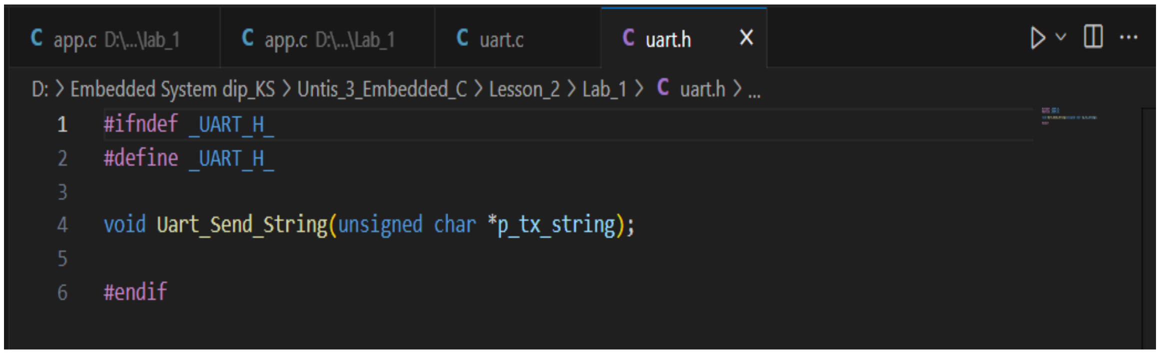
➢ uart.c



app.c D:\...\lab_1
app.c D:\...\Lab_1
uart.c
D: > Embedded System dip_KS > Untis_3_Embedded_C > Lesson_2 > Lab_1 > uart.c > ...

```c
#include "uart.h"
#define UART0DR *(volatile unsigned int *)(unsigned int*)0x101f1000

void Uart_Send_String(unsigned char* p_tx_string )
{
    while(*p_tx_string!='\0')
    {
        UART0DR=(unsigned int)(*p_tx_string);
        p_tx_string++;
    }
}
```

➢ uart.h



app.c D:\...\lab_1
app.c D:\...\Lab_1
uart.c
uart.h
D: > Embedded System dip_KS > Untis_3_Embedded_C > Lesson_2 > Lab_1 > uart.h > ...

```c
#ifndef _UART_H_
#define _UART_H_

void Uart_Send_String(unsigned char *p_tx_string);

#endif
```