

arm-gcc++ Compiler

Fernando Simplicio de Sousa

O **GNU Compiler Collection** (chamado usualmente por **GCC**) é um conjunto de compiladores de linguagens de programação.

Arm-gcc++ Compiler

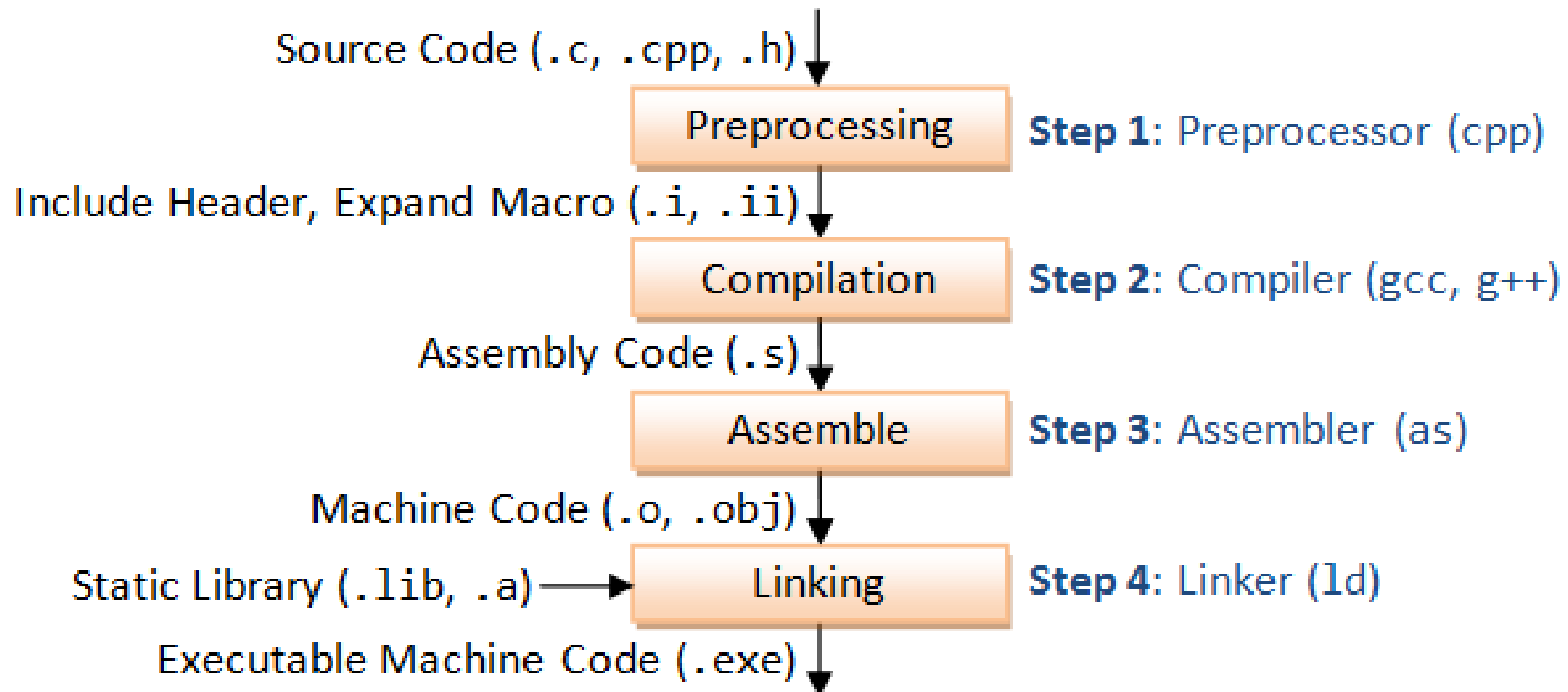
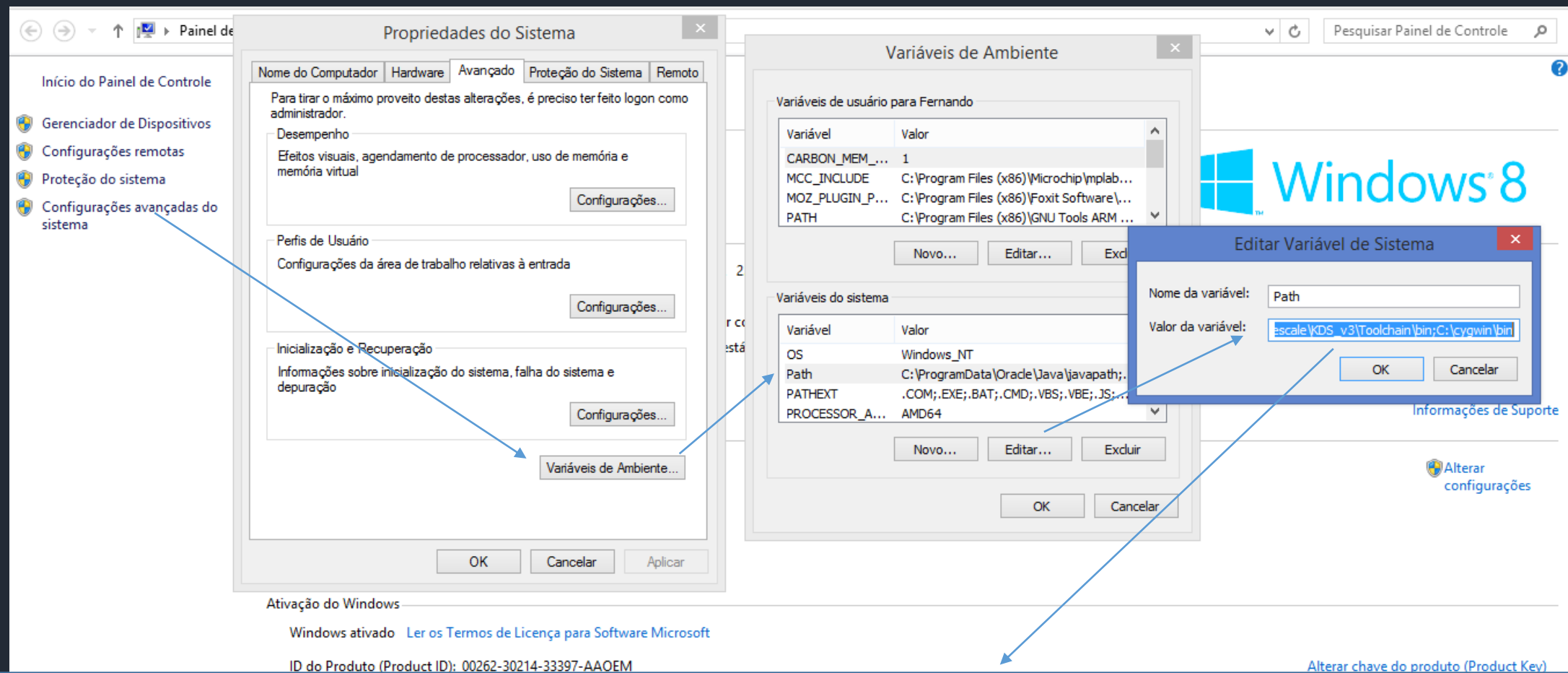


Imagem: https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

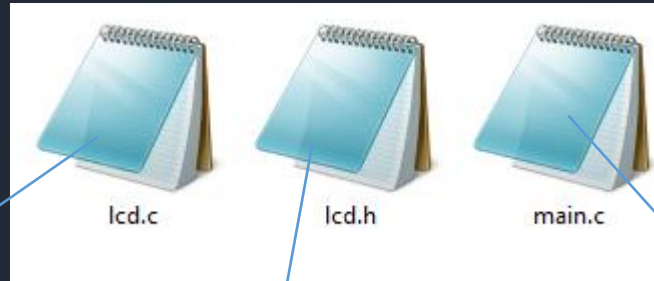
avr-gcc++

2º Configurando as variáveis de ambiente no Windows.



C:\Program Files (x86)\Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain\bin

avr-gcc++ Compiler (Exemplo)



```
#include <stdio.h>
#include "lcd.h"

void lcd_init()
{
    printf("BOA TARDE!\r");
}
```

```
#ifndef __LCD_INIT_H__
#define __LCD_INIT_H__

void lcd_init();

#endif
```

```
#include <stdio.h>
#include "lcd.h"

int main()
{
    printf("BOM DIA!\r");
    return 0;
}
```

avr-gcc++ Compiler

```
$ avr-gcc -c -O output.o main.c lcd.c -I.
```

\$ -> Prompt de Comandos.

gcc -> Compilador.

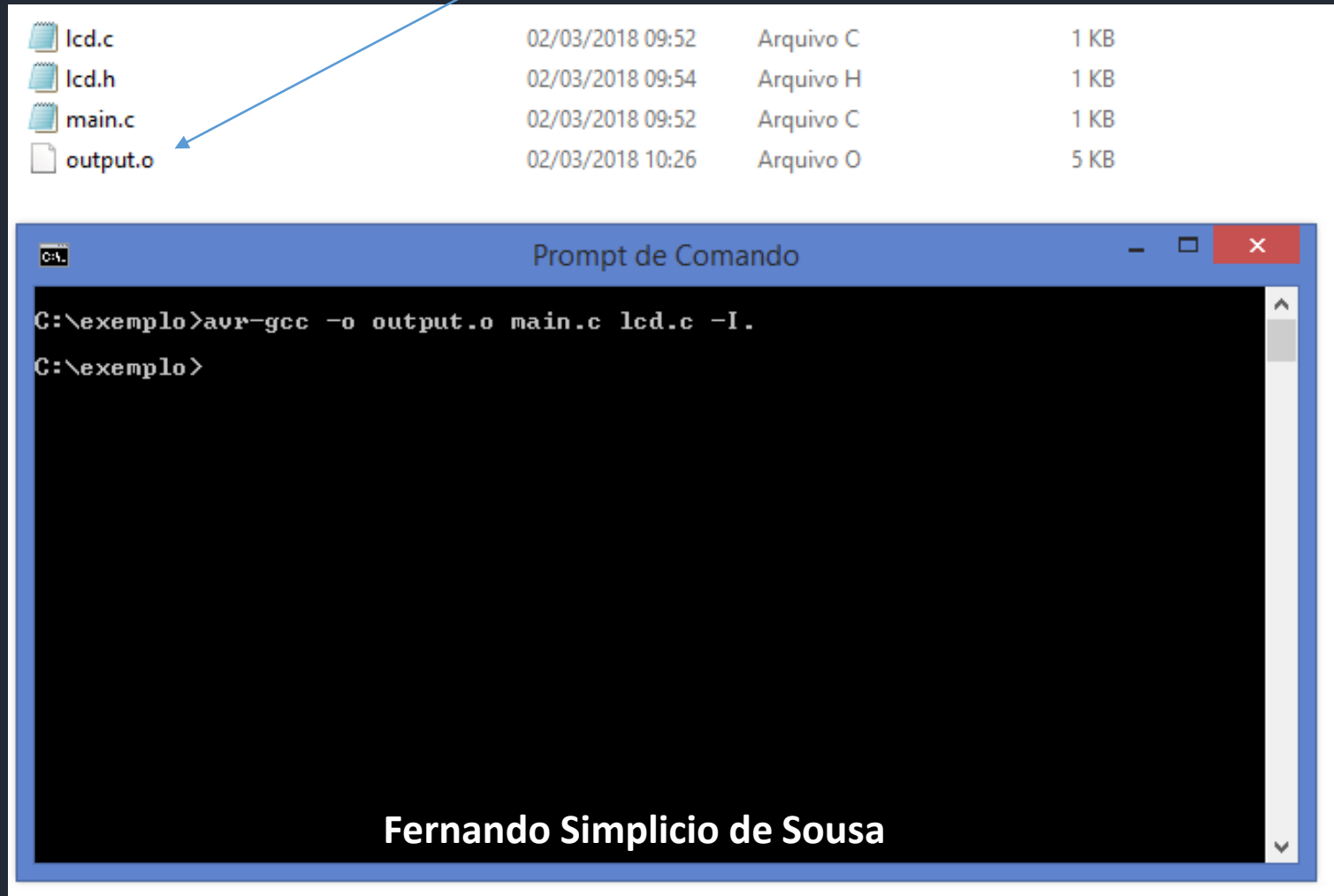
-c -> Gerar um objeto (*.o) na saída.

-O -> Arquivo a ser criado na saída.

-I. -> Busca os arquivos .c .h no diretório raiz.

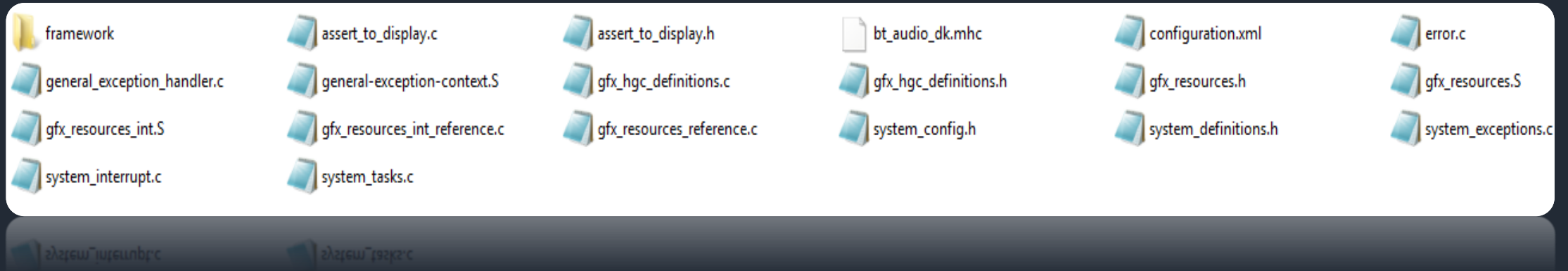
avr-gcc++ Compiler

```
$ avr-gcc -o output.o main.c lcd.c -I.
```



Ok, mas...

Como compilar esse projeto?!



Arquivo Makefile

O makefile é um arquivo para configuração de compilação utilizado pelo programa Make, cuja idéia é simplificar e agilizar a compilação de programas [1].

```
include ../makefile.conf
NAME=semihost
STARTUP_DEFS=

LDSCRIPTS=-L. -L$(BASE)/ldscripts -T gcc.ld
LFLAGS=$(USE_NANO) $(USE_SEMIHOST) $(LDSCRIPTS) $(GC) $(MAP)

$(NAME)-$(CORE).axf: $(NAME).c $(STARTUP)
    $(CC) $^ $(CFLAGS) $(LFLAGS) -o $@

clean:
    rm -f $(NAME)*.axf *.map
```

Dependências

Antes de executar uma regra, o programa **make** se certifica que todas as dependências foram supridas.

Regra: dependências

comando1

comando2

comando3

comando4

Programa Make

Os comandos devem ser identados por meio de TAB



```
Regra: dependências  
[TAB] comando1  
[TAB] comando2  
[TAB] comando3  
[TAB] comando4
```

Programa Make

O programa **make** verifica se há necessidade de recompilar um arquivo por meio do nome do arquivo.

```
main.hex: main.c main.h  
    g++ -o main.hex main.c
```

Programa Make

O programa **make** verifica as dependências das regras.

```
upload: main.hex  
    main.hex
```

```
main.hex: main.c main.h  
    g++ -o main.hex main.h
```

Exemplo makefile

A ordem de execução do programa **make** segue a enumeração

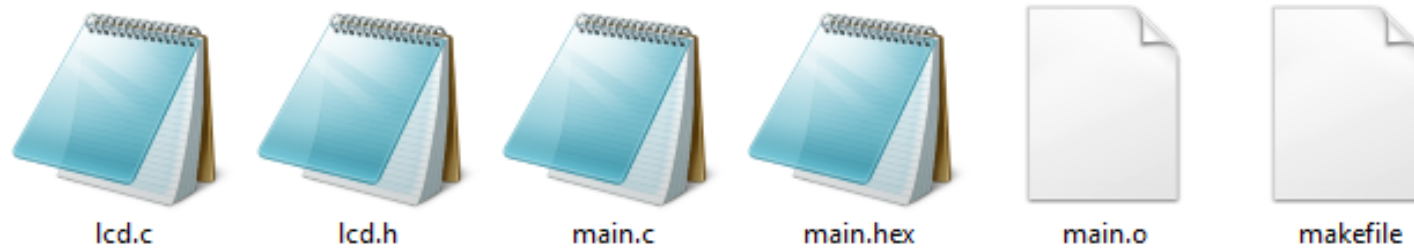
```
all: msg main.hex
    @echo '4- Compilacao finalizada com sucesso!'
    @echo '5- Gravando mcu...'

msg:
    @echo '1- Inicio de Compilacao do Programa.'

main.hex: main.o
    @echo '3- Gerando o arquivo main.hex'
    avr-gcc -o main.hex main.o

main.o: main.c lcd.c lcd.h
    @echo '2- Gerando o arquivo main.o'
    avr-gcc -c main.c
```

Resultado após executar o programa make



```
C:\proggcc3>make
1- Inicio de Compilacao do Programa.
2- Gerando o arquivo main.o
avr-gcc -c main.c
3- Gerando o arquivo main.hex
avr-gcc -o main.hex main.o
4- Compilacao finalizada com sucesso!
5- Gravando mcu...

C:\proggcc3>_
```

Exemplo makefile

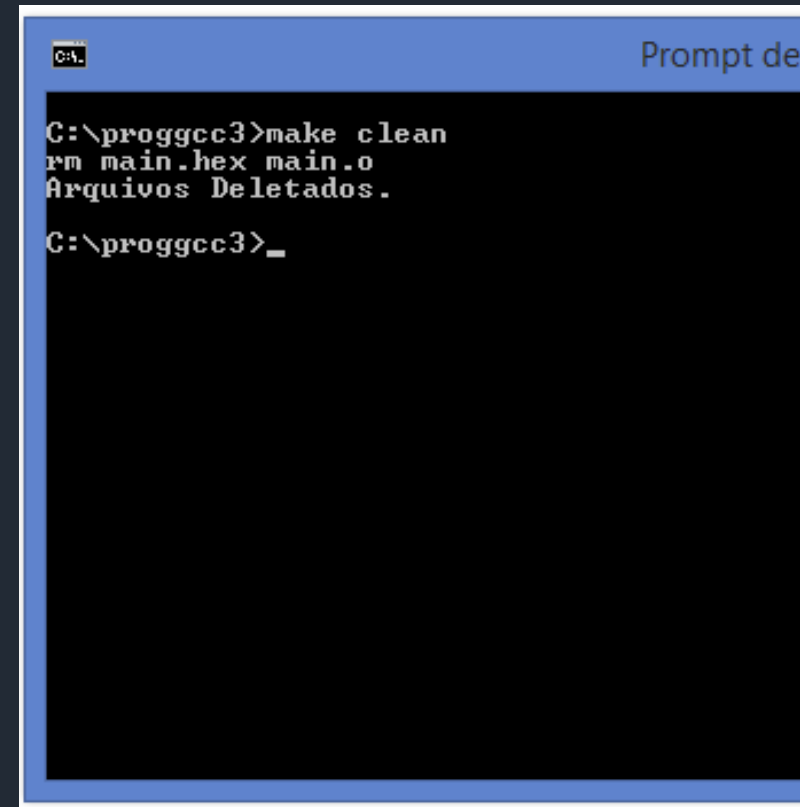
Caso o arquivo main.hex exista, o programa **make** não irá recompilar novamente o código. Portanto, podemos adicionar um comando para apagar os arquivos e assim gerar o main.hex novamente.

```
msg:
    @echo '1- Inicio de Compilacao do Programa.'

main.hex: main.o
    @echo '3- Gerando o arquivo main.hex'
    avr-gcc -o main.hex main.o

main.o: main.c lcd.c lcd.h
    @echo '2- Gerando o arquivo main.o'
    avr-gcc -c main.c

clean:
    rm main.hex main.o
    @echo 'Arquivos Deletados.'
```

A screenshot of a terminal window with a blue title bar. The title bar contains a small icon and the text 'Prompt de'. The terminal shows the command 'C:\proggcc3>make clean' being executed. The output is 'rm main.hex main.o' followed by 'Arquivos Deletados.' on the next line. The prompt 'C:\proggcc3>' appears again on the following line. A blue arrow points from the 'clean:' target in the Makefile on the left to the 'make clean' command in the terminal.

```
C:\proggcc3>make clean
rm main.hex main.o
Arquivos Deletados.
C:\proggcc3>
```


Variáveis no Arquivo makefile

É possível adicionar variáveis no arquivo makefile para facilitar as construções das regras e geração das saídas pelo programa **maker**.

```
# Selecting Core
CORTEX_M=0

# Use newlib-nano. To disable it, specify USE_NANO=
USE_NANO=--specs=nano.specs

# Use seimhosting or not
USE_SEMIHOST=--specs=rdimon.specs
USE_NOHOST=--specs=nosys.specs

CORE=CM$(CORTEX_M)
BASE=../..

# Compiler & Linker
CC=arm-none-eabi-gcc
CXX=arm-none-eabi-g++

# Options for specific architecture
ARCH_FLAGS=-mthumb -mcpu=cortex-m$(CORTEX_M)
```

Exemplo-1 makefile

CC Variável

CFLAGS Variável

%.o %.c %.h Faz referência a extensão do arquivo de chamada

\$< Corresponde ao nome da primeira dependência, neste caso, main.c, lcd.c

\$@ Nome do objeto “alvo” que será criado.

```
CC=avr-gcc
CFLAGS=-I.

all: msg main.hex
    @echo '4- Compilacao finalizada com sucesso!'
    @echo '5- Gravando mcu...'

msg:
    @echo '1- Inicio de Compilacao do Programa.'

main.hex: main.o lcd.o
    @echo '3- Gerando o arquivo main.hex'
    $(CC) -o main.hex main.o $(CFLAGS)

%.o: %.c %.h
    @echo '2- Gerando o arquivo main.o'
    $(CC) -c $< -o $@ $(CFLAGS)

clean:
    rm main.hex main.o lcd.o
    @echo 'Arquivos Deletados.'
```

Exemplo-2 makefile

SRCS Relação dos arquivos Sources .c

CFLAGS Variável

%.o %.c %.h Faz referência a extensão do arquivo de chamada

\$< Corresponde ao nome da primeira dependência, neste caso, main.c, lcd.c

\$@ Nome do objeto “alvo” que será criado.

```
CC=avr-gcc
CFLAGS=-I.
SRCS=main.c \
      lcd.c \

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

all: main.hex

main.hex:$(OBJS)
      $(CC) $(OBJS) -o $@

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

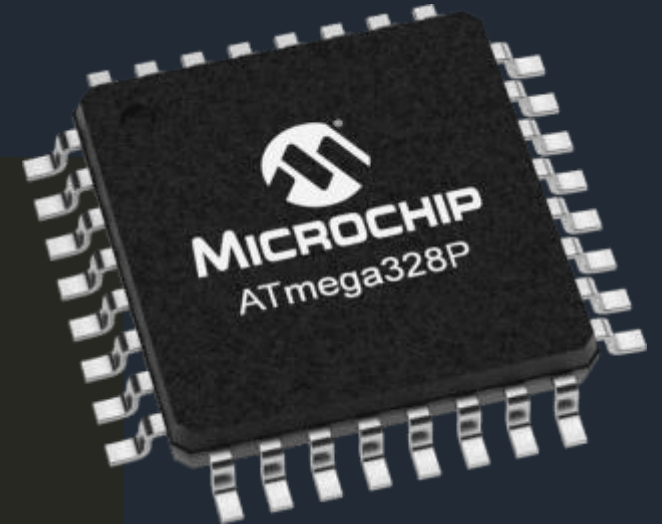
clean:
      rm -f $(OBJS)
```

Exemplo Makefile para AVR-MCU



main.c

```
1  #define F_CPU 16000000UL    // Define a frequência do microcontrolador
2  #include <avr/io.h>         // Biblioteca para o uso dos I/Os do MCU
3  #include <util/delay.h>     // Biblioteca para o uso das rotinas de delay
4
5  void ConfigMCU()
6  {
7      DDRB = 0xFF;            //Configura todos os pinos do PORTB como saída
8      PORTB = 0;              //Todo o PORTB terá nível lógico zero.
9  }
10
11 int main(void)
12 {
13     ConfigMCU();
14     while (1)
15     {
16         PORTB ^= 0xFF;       //Toggle, ou seja, inverte dos os pinos do PORTB;
17         _delay_ms(300);      //Delay de 300ms
18     }
19 }
```



Exemplo gcc++ para AVR-MCU



1º compilar o arquivo

```
$ avr-gcc -g -Os -mmcu=atmega328p -c main.c
```

\$ -> Prompt de Comandos.

avr-gcc -> Compilador.

-g -> (Opcional) Geração de informações de debug (útil quando for desmontar o código).

-Os -> (Opcional) Geração de códigos mais otimizados.

-mmcu. -> Informa qual o microcontrolador alvo.

Exemplo gcc++ para AVR-MCU



2° gerar o arquivo binário *.elf

```
$ avr-gcc -g -mmcu=atmega328p -o main.elf main.o
```

Para a geração do arquivo binário *.elf devemos carregar o arquivo objeto main.o e informar novamente qual o microcontrolador alvo. Caso não seja especificado o microcontrolador, será considerado o chip padrão AVR90S8515

Exemplo gcc++ para AVR-MCU



3° Examinar a geração dos arquivos.

```
$ avr-objdump -h -S main.elf > main.lst
```

Dentro do pacote **GNU Binutils *suíte*** do gcc encontramos o programa objdump. Por meio deste programa é possível obter informações sobre os arquivos objetos e apresentá-las de muitas formas.

- h (opcional) Informa o tamanho do programa.
- S Esta opção desmonta o arquivo binário e intercala o código fonte na saída!

Exemplo gcc++ para AVR-MCU



4º Geração do Arquivo *.map

```
$ avr-gcc -g -mmcu=atmega328p -Wl,-Map=main.map  
-o main.elf main.o
```

O arquivo *.map contém o reports da compilação do programa, e informa o consumo de memória e quais os módulos e bibliotecas foram carregados no programa.

-Wl (opcional) Permite carregar atributos para o linker.

-S Esta opção desmonta o arquivo binário e intercala o código fonte na saída!

-g: gera informações de debugging simbólicas adicionais para uso com o depurador gdb.

Exemplo gcc++ para AVR-MCU



4° Convertendo o arquivo *.elf para *.hex

```
$ avr-objcopy -j .text -j .data -O ihex main.elf  
main.hex
```

Nem todos os gravadores suportam o padrão de arquivo binário *.elf (padrão GNU). Portanto, em muitos casos é necessário convertê-lo para o padrão *.hex (intel). O programa objcopy do gcc é quem faz essa conversão.

Exemplo gcc++ para AVR-MCU



5° Gerar o arquivo EEPROM.hex (em caso de uso)

```
$ avr-objcopy -j .eeprom --change-section-lma  
.eeprom=0 -O ihex main.elf main_eeprom.hex
```

Por meio do programa objcopy do gcc é possível extrair do binário *.elf os dados do setor da EEPROM e salvar (em hexadecimal) os dados em um outro arquivo.

-J Informa que queremos as informações em .text e .data.

Atenção: Este arquivo somente será gerado caso exista alguma informação no setor da eeprom.

Arquivo *.map

.text

Consumo de memória Flash (alocação das instruções)

.text **0x00000000**
_etext = **0x000000ac**

Total: 172 bytes

```
.text            0x00000000       0xac  
*(.vectors)  
.vectors       0x00000000       0x68  
                 0x00000000  
                 0x00000000  
*(.vectors)  
*(.progmem.gcc*)  
*(.progmem*)  
                 0x00000068  
                 0x00000068  
*(.trampolines)  
.trampolines   0x00000068       0x0  
*(.trampolines*)  
                 0x00000068  
*(.trampolines*)
```

Arquivo *.data e bss

.data e bss

Consumo de memória RAM para variáveis inicializadas (data) e não inicializadas (bss).

Ambos setores inicializados a partir do endereço 100.
(logo após os bancos de registradores).

Tamanhos: 0 byte.

```
.data          0x00800100      0x0 load address 0x000000ac
               0x00800100      PROVIDE (__data_start, .)

*(.data)
.data          0x00800100      0x0 c:/winavr-20100110/bin/../../lib
.data          0x00800100      0x0 main.o
.data          0x00800100      0x0 c:/winavr-20100110/bin/../../lib
*(.data*)
*(.rodata)
*(.rodata*)
*(.gnu.linkonce.d*)
               0x00800100      . = ALIGN (0x2)
               0x00800100      _edata = .
               0x00800100      PROVIDE (__data_end, .)

.bss           0x00800100      0x0
               0x00800100      PROVIDE (__bss_start, .)

*(.bss)
.bss           0x00800100      0x0 c:/winavr-20100110/bin/../../lib
.bss           0x00800100      0x0 main.o
.bss           0x00800100      0x0 c:/winavr-20100110/bin/../../lib
*(.bss*)
*(COMMON)
               0x00800100      PROVIDE (__bss_end, .)
```

Arquivo *.eeprom

.eeprom

Consumo de memória EEPROM.

Inicializada no endereço 0.

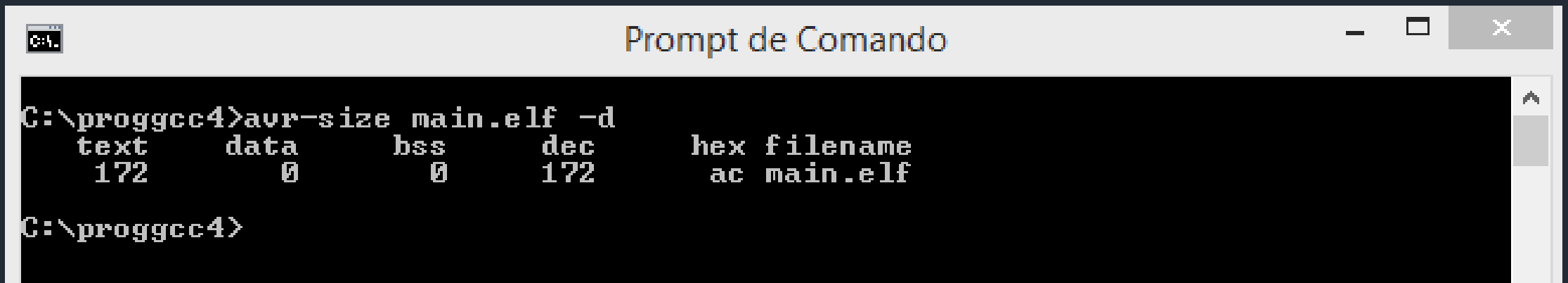
Tamanhos: 0 byte.

```
.eeprom          0x00810000      0x0  
*(.eeprom*)  
                0x00810000      __eeprom_end = .
```

gcc Reporter

É possível verificar o consume de memória diretamente através do programa `avr-size/arm-gcc-size` do gcc.

```
$ avr-size main.elf -d
```



```
C:\proggcc4>avr-size main.elf -d
```

text	data	bss	dec	hex	filename
172	0	0	172	ac	main.elf

```
C:\proggcc4>
```

Hex vs Elf (diferenças)

Elf (*Executable e Linking Format*) é um formato padrão comum para arquivos executáveis, código de objeto e bibliotecas compartilhadas. É criado pelo **gcc** como o resultado da compilação.

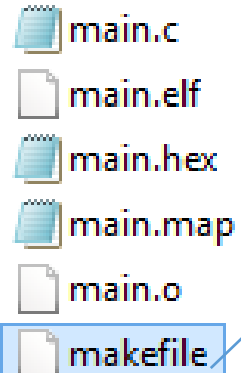
O programa **objcopy do gcc** é usado para converter o arquivo ***.elf** para o formato de arquivo hexadecimal que é um arquivo de texto hexadecimal padronizado usado para programar o dispositivo AVR/ARM.

Exercício (individual)

- 1- Compilar o programa **aula.c** e gerar o arquivo *.elf.
- 2- Converter o arquivo *.elf para o format hexadecimal *.hex.
- 3- Gerar o arquivo *.map e verificar a configuração de memória do programa.

**Simplificação do fluxo de compilação
por meio do arquivo **makefile****

Solução



main.c
main.elf
main.hex
main.map
main.o
makefile

```
#
CC=avr-gcc
CFLAGS= -g -Os -mmcu=atmega328p
LDFLAGS= -Wl,-Map=main.map
OBJCOPY=avr-objcopy
OBJDUMP=avr-objdump

all: main.hex main.elf

main.elf: main.o
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^

main.o: main.c
    $(CC) $(CFLAGS) -c $< -o $@

main.lst: main.elf
    $(OBJDUMP) -d $< > $@

main.hex: main.elf
    $(OBJCOPY) -j .text -j .data -O ihex $< $@

program: main.hex
    avrdude -p atmega328p -c arduino -P COM14 -b 115200 -U flash:w:main.hex:i

clean:
    rm -rf *.o *.elf
    rm -rf *.lst *.map
```

Solução

main.c
main.elf
main.hex
main.map
main.o
makefile

```
C:\>
C:\proggcc4>make
avr-gcc -g -Os -mmcu=atmega328p -c main.c -o main.o
avr-gcc -g -Os -mmcu=atmega328p -Wl,-Map=main.map -o main.elf main.o
avr-objcopy -j .text -j .data -j .ihex main.elf main.hex

C:\proggcc4>make program
avrdude -p atmega328p -c arduino -P COM14 -b 115200 -U flash:w:main.hex:i

avrdude: AVR device initialized and ready to accept instructions

Reading : ##### : 100% 0.01s

avrdude: Device signature = 0x1e950f
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed

        To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: writing flash (172 bytes):

Writing : ##### : 100% 0.06s

avrdude: 172 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 172 bytes
avrdude: reading on-chip flash data:

Reading : ##### : 100% 0.04s

avrdude: verifying ...
```

arm-gcc++ e KL25Z

arm-gcc++ e KL25Z

1º Fazer download do programa GNU Embedded Toolchain (arm-gcc++) em

<https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads>

**Nota: O programa arm-gcc++ vem junto com a IDE
IDE Kinetis Studio !**

arm-gcc++ e KL25Z

2º Configurando as variáveis de ambiente no Windows.

The screenshot illustrates the steps to configure environment variables in Windows 8:

- System Properties:** The 'Advanced' tab is selected, and the 'Environment Variables...' button is clicked.
- Environment Variables:** This window shows two sections:
 - Variáveis de usuário para Fernando:** A table with columns 'Variável' and 'Valor'. It lists:

Variável	Valor
CARBON_MEM_...	1
MCC_INCLUDE	C:\Program Files (x86)\Microchip\mplab...
MOZ_PLUGIN_P...	C:\Program Files (x86)\Foxit Software\...
PATH	C:\Program Files (x86)\GNU Tools ARM ...
 - Variáveis do sistema:** A table with columns 'Variável' and 'Valor'. It lists:

Variável	Valor
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;...
PROCESSOR_A...	AMD64
- Edit System Variable:** The 'Path' variable is selected, and this dialog is used to edit its value. The 'Nome da variável' is 'Path', and the 'Valor da variável' is being updated to include 'C:\Freescale\KDS_v3\Toolchain\bin'.

Ativação do Windows

Windows ativado [Ler os Termos de Licença para Software Microsoft](#)

ID do Produto (Product ID): 00262-30214-33397-AAOEM

Consulte também
Central de Ações
Windows Update

C:\Freescale\KDS_v3\Toolchain\bin

Alterar chave do produto (Product Key)

arm-gcc++ e KL25Z

3° Verifica se a variável de ambiente foi corretamente referenciada no computador.

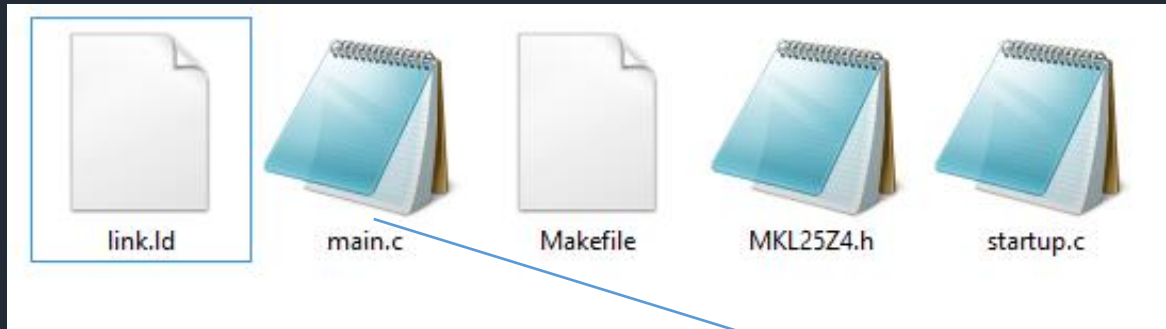


Prompt de Comando

```
C:\proggcc>arm-none-eabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-none-eabi-gcc
COLLECT_LTO_WRAPPER=c:/freescale/kds_v3/toolchain/bin/./lib/gcc/arm-none-eabi/4
.8.4/lto-wrapper.exe
Target: arm-none-eabi
Configured with: /home/build/work/GCC-4-8-build/src/gcc/configure --build=i686-l
inux-gnu --host=i686-w64-mingw32 --target=arm-none-eabi --prefix=/home/build/wor
k/GCC-4-8-build/install-mingw --libexecdir=/home/build/work/GCC-4-8-build/instal
l-mingw/lib --infodir=/home/build/work/GCC-4-8-build/install-mingw/share/doc/gcc
-arm-none-eabi/info --mandir=/home/build/work/GCC-4-8-build/install-mingw/share/
doc/gcc-arm-none-eabi/man --htmldir=/home/build/work/GCC-4-8-build/install-mingw
/share/doc/gcc-arm-none-eabi/html --pdfdir=/home/build/work/GCC-4-8-build/instal
l-mingw/share/doc/gcc-arm-none-eabi/pdf --enable-languages=c,c++ --disable-decim
J-wtubdm\sgprte\qoc\acc-gnw-uoue-egprt\bqf --egprtj6-J9uAn9des=c`c++ --qiz9prtj6-qecrw
\sgprte\qoc\acc-gnw-uoue-egprt\pfcwJ --bqfqtz=\powe\prttjq\woyk\GCC-4-8-prttjq\tuscf9J
qoc\acc-gnw-uoue-egprt\w9u --pfcwJqtz=\powe\prttjq\woyk\GCC-4-8-prttjq\tuscf9JJ-wtubdm
```

arm-gcc++ e KL25Z

4° Carregar o projeto para a KL25Z conforme figura.



```
main.c x
1  #include "MKL25Z4.h"
2
3  volatile unsigned int t = 10;
4  void delay(void) {
5      int i;
6      for (i = 0; i < 1000000; i++) {
7          }
8      }
9
10 void toggle_red(void) {
11     GPIOB_PTOR = (1 << 18);
12 }
13
14 void toggle_green(void) {
15     GPIOB_PTOR = (1 << 19);
16 }
17
18 void toggle_blue(void) {
19     GPIOD_PTOR = (1 << 1);
20 }
21
22 void main(void) {
23     /* disable COP */
```



```

1 ARCHFLAGS=-mthumb -mcpu=cortex-m0plus
2 CFLAGS=
3 LDFLAGS=--specs=nano.specs -Wl,--gc-sections,-Map,$(TARGET).map,-T link.ld
4
5 CC=arm-none-eabi-gcc
6 LD=arm-none-eabi-gcc
7 OBJCOPY=arm-none-eabi-objcopy
8 SIZE=arm-none-eabi-size
9 RM=rm -f
10
11 TARGET=main
12 SRC=$(wildcard *.c)
13 OBJ=$(patsubst %.c, %.o, $(SRC))
14
15 all: build size
16 build: elf srec
17 elf: $(TARGET).elf
18 srec: $(TARGET).srec
19
20 clean:
21     $(RM) $(TARGET).srec $(TARGET).elf $(TARGET).map $(OBJ)
22
23 %.o: %.c
24     $(CC) -c $(ARCHFLAGS) $(CFLAGS) -o $@ $<
25
26 $(TARGET).elf: $(OBJ)
27     $(LD) $(ARCHFLAGS) $(LDFLAGS) -o $@ $(OBJ)
28
29 %.srec: %.elf
30     $(OBJCOPY) -O srec $< $@
31
32 size:
33     $(SIZE) $(TARGET).elf
34

```

arm-gcc++ makefile



arm-gcc++ e KL25Z

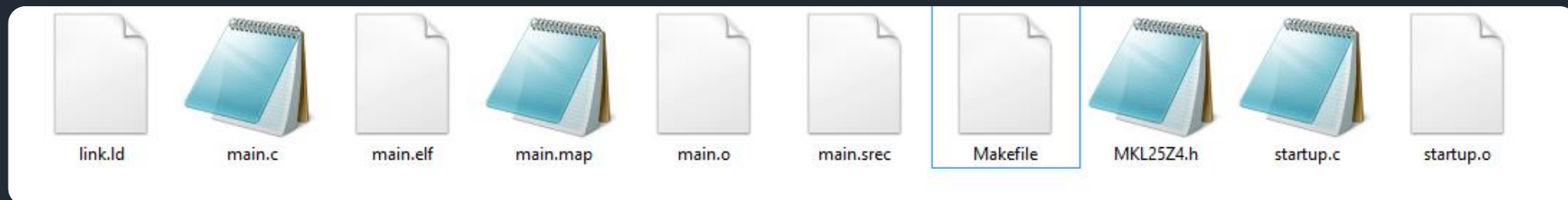
5° Compila o projeto por meio do programa make



```
C:\proggcc>make
arm-none-eabi-gcc -c -mthumb -mcpu=cortex-m0plus -o main.o main.c
arm-none-eabi-gcc -c -mthumb -mcpu=cortex-m0plus -o startup.o startup.c
arm-none-eabi-gcc -mthumb -mcpu=cortex-m0plus --specs=nano.specs -Wl,--gc-sections,-Map,main.map,-T link.ld -o main.elf main.o startup.o
arm-none-eabi-objcopy -O srec main.elf main.srec
arm-none-eabi-size main.elf
   text    data     bss     dec     hex filename
   592       0    1024    1616     650 main.elf

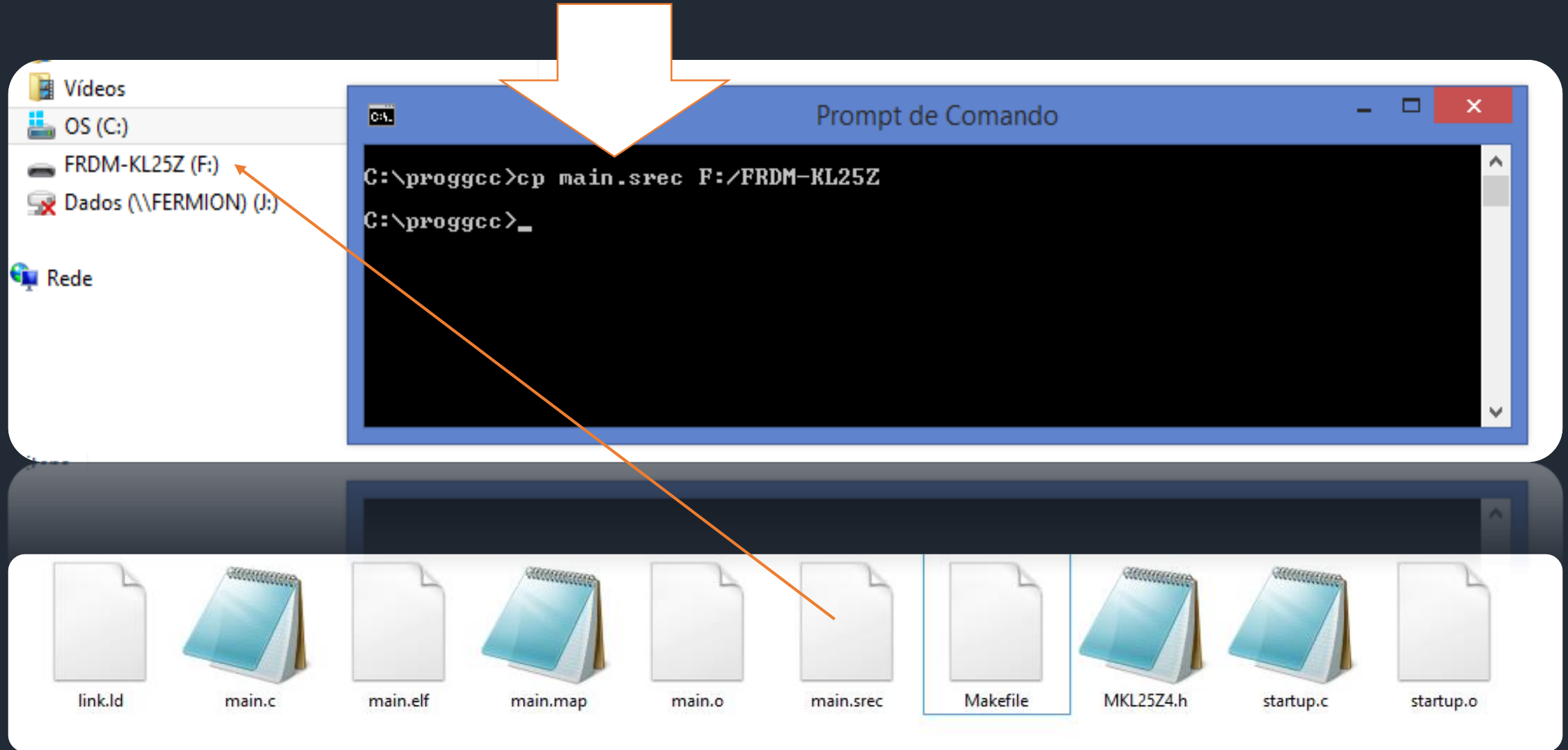
C:\proggcc>_
```

Resultado:



arm-gcc++ e KL25Z

6° Faz a gravação do programa na KL25Z via OpenSDA



Exercício (individual)

1- Compile o projeto (**aula2.c**) e verifique o consume de memória (.text, bss, data) por meio do programa **arm-none-eabi-size**.

```
$ arm-none-eabi-size --format=sysv main.srec
```

Exercício (individual)

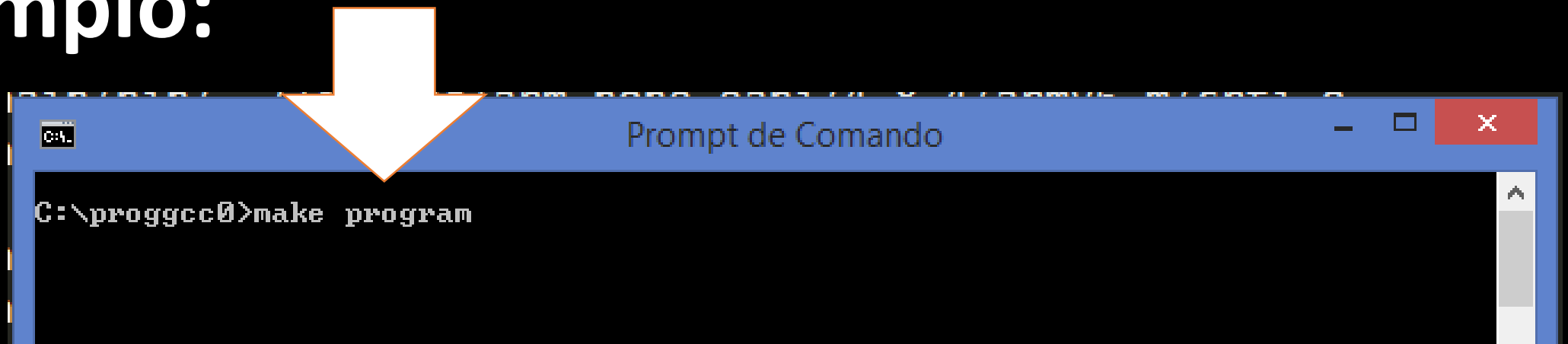
2- Adicione as seguintes variáveis GLOBALS no programa, recompile e verifique o consume de memória em .text, .bss e data.

```
volatile int32_t myVar = 0x12345678;  
const int table[] = {5,0,1,5,6,7,9,10};
```

Exercício (individual)

3- Adicione no arquivo makefile do projeto (**aula2.c**) uma nova regra para a realização da gravação do **main.srec** na KL25Z.

Exemplo:



```
C:\proggcc0>make program
```

Importantes links sobre o gcc++

https://www.microchip.com/webdoc/AVRLibcReferenceManual/using_tools_1using_sel_gcc_opts.html#using_tools_1gcc_minusW

<https://sourceware.org/binutils/docs-2.27/binutils/>

https://pt.wikibooks.org/wiki/Programar_em_C/Makefiles

https://www.ntu.edu.sg/home/ehchua/programming/howto/Cygwin_HowTo.html

<https://terminaldeinformacao.com/2015/10/08/como-instalar-e-configurar-o-gcc-no-windows-mingw/>

<http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>

<https://pt.coursera.org/learn/introduction-embedded-systems/lecture/kdeCy/8-makefiles-part-2>

https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

Obrigado