

Sistemas Embarcados

`GCC e make`

Compilação de códigos em C no Linux

- Para desenvolvimento de códigos em C no Linux, utilizaremos o “GCC” em conjunto com o “make”
 - O “GCC” é responsável por:
 - Compilar o código em C, gerando o código de máquina correspondente.
 - Ligar códigos de máquina separados em um único programa executável.
 - O “make” é responsável por definir as regras de compilação do código, facilitando o trabalho de desenvolvimento

GCC

- Considere este código:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("Ola mundo!\n");
    return 0;
}
```

GCC

- Considere este código:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("Ola mundo!\n");
    return 0;
}
```

- Considere que ele foi salvo em “~/Code/03_GCC/Ex0/main.c”:

```
~/Code/03_GCC/Ex0/ $ ls
main.c
```

GCC

- Considere este código:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("Ola mundo!\n");
    return 0;
}
```

Para o GCC compilar este código, gerando um programa executável chamado “ola_mundo”, digitamos:

```
~/Code/03_GCC/Ex0/ $ gcc main.c -o ola_mundo
~/Code/03_GCC/Ex0/ $ ls -l
-rw-rw-r-- 1 diogo diogo 100 mar 18 14:33 main.c
-rwxr-xr-x 1 diogo diogo 8296 mar 18 14:42 ola_mundo
```

GCC

- Considere este código:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("Ola mundo!\n");
    return 0;
}
```

Para executar este novo programa, digitamos:

```
~/Code/03_GCC/Ex0/ $ ./ola_mundo
Ola mundo!
```

GCC

- Considere este código:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("Ola mundo!\n");
    return 0;
}
```

Neste exemplo simples, a função “main()” recebeu dois parâmetros de entrada, “argc” e “argv”. Vejamos o que eles representam.

GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

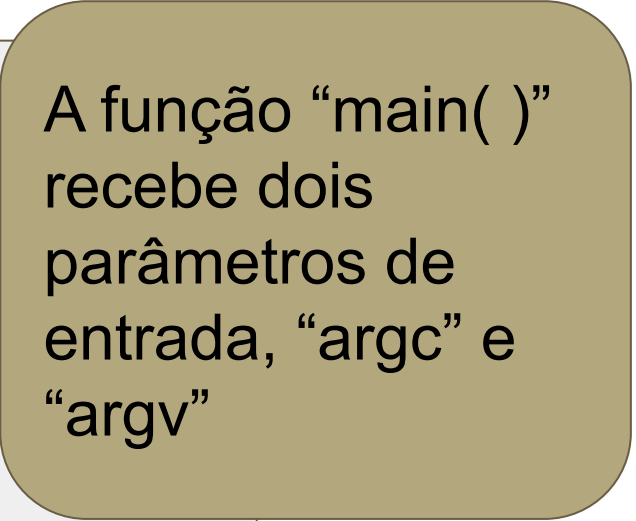

GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

A função “main()”
recebe dois
parâmetros de
entrada, “argc” e
“argv”



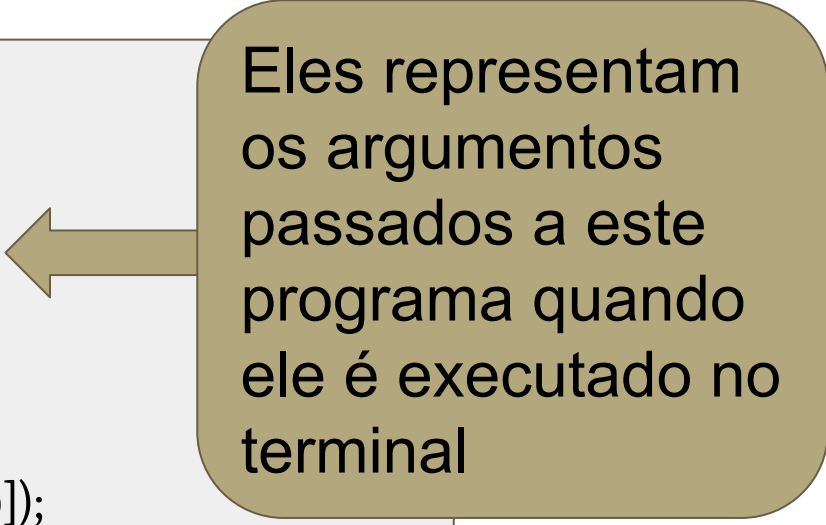
GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

Eles representam
os argumentos
passados a este
programa quando
ele é executado no
terminal



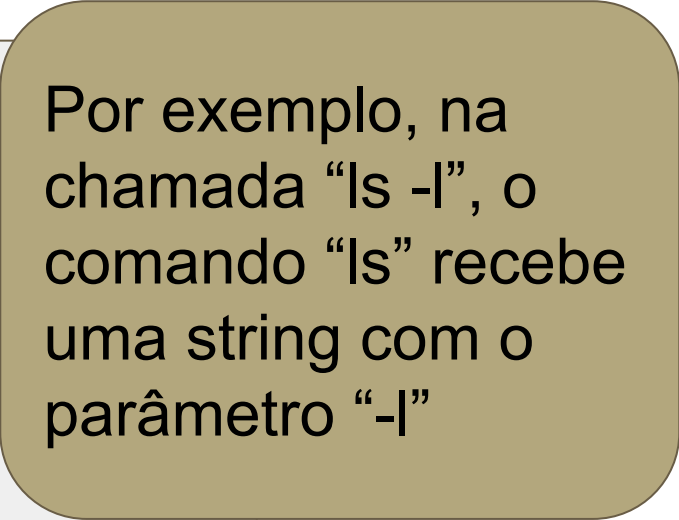
GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

Por exemplo, na chamada “ls -l”, o comando “ls” recebe uma string com o parâmetro “-l”



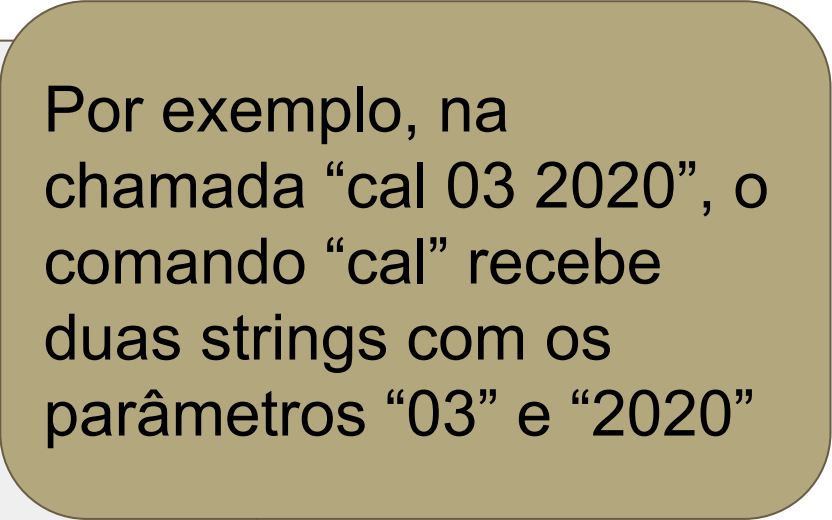
GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

Por exemplo, na chamada “cal 03 2020”, o comando “cal” recebe duas strings com os parâmetros “03” e “2020”



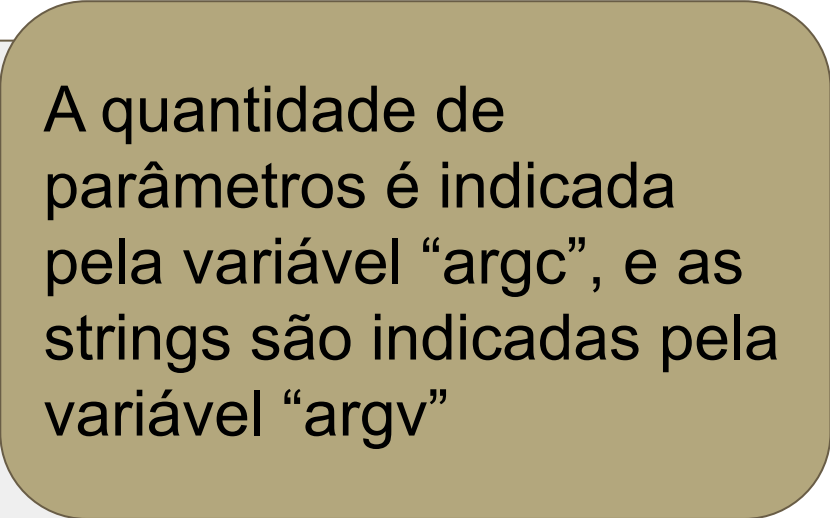
GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

A quantidade de parâmetros é indicada pela variável “argc”, e as strings são indicadas pela variável “argv”



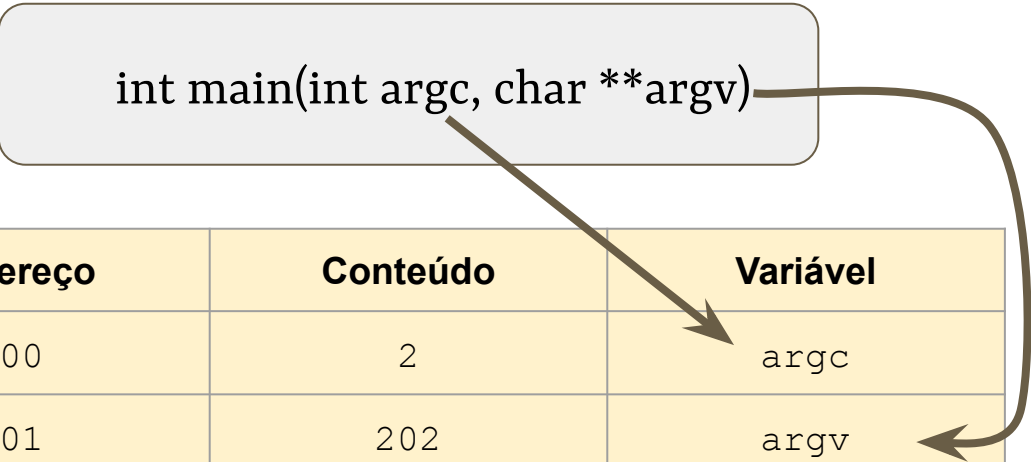
Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```

Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	argv[1]
204	'l'	argv[0][0]
205	's'	argv[0][1]
206	'\0'	argv[0][2]
207	'-'	argv[1][0]
208	'l'	argv[1][1]
209	'\0'	argv[1][2]

Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```



Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	argv[1]
204	'l'	argv[0][0]
205	's'	argv[0][1]
206	'\0'	argv[0][2]
207	'-'	argv[1][0]
208	'l'	argv[1][1]
209	'\0'	argv[1][2]

Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```

Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	argv[1]
204	'l'	argv[0][0]
205	's'	argv[0][1]
206	'\0'	argv[0][2]
207	'-'	argv[1][0]
208	'l'	argv[1][1]
209	'\0'	argv[1][2]

“argc = 2” significa
que o usuário
passou dois
parâmetros

Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```

Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	
204	'l'	<p>Primeiro parâmetro ("ls")</p>
205	's'	
206	'\0'	
207	'-'	
208	'l'	argv[1][1]
209	'\0'	argv[1][2]

Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```

Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	argv[1]
204	'l'	argv[0][0]
205	's'	argv[0][1]
206	'\0'	
207	'-'	
208	'l'	
209	'\0'	

Segundo
parâmetro
("-l")

Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```

Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	argv[1]
204	'l'	argv[0][0]
205	's'	argv[0][1]
206	'\0'	argv[0][2]
207	'-'	argv[1][0]
208	'l'	argv[1][1]
209	'\0'	argv[1][2]

Como são dois parâmetros, e eles chegam através de strings, precisamos saber onde estão estas duas strings.

Em C, as strings são salvas em vetores, logo precisamos de um vetor de vetores

“argv” é um vetor de vetores, que é o mesmo que um ponteiro para ponteiros

Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```

Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	argv[1]
204	'l'	argv[0][0]
205	's'	argv[0][1]
206	'\0'	argv[0][2]
207	'-'	argv[1][0]
208	'l'	argv[1][1]
209	'\0'	argv[1][2]

“argv[0]” indica o endereço do primeiro parâmetro

Por exemplo, na chamada “ls -l”:

```
int main(int argc, char **argv)
```

Endereço	Conteúdo	Variável
200	2	argc
201	202	argv
202	204	argv[0]
203	207	argv[1]
204	'l'	argv[0][0]
205	's'	argv[0][1]
206	'\0'	argv[0][2]
207	'-'	argv[1][0]
208	'l'	argv[1][1]
209	'\0'	argv[1][2]

“argv[1]” indica o endereço do segundo parâmetro


GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

Voltando ao código, esta linha
imprime a quantidade de
parâmetros passados pelo
usuário a este programa



GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

Pelo menos um parâmetro é garantido: o próprio nome do programa compilado.

Esta linha imprime “argv[0]”, que é o nome deste programa

GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

Se o usuário passou mais de um parâmetro, então o programa imprime "argv[1]", que é o segundo parâmetro passado

GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

Se o usuário passou mais de dois parâmetros, então o programa imprime “argv[2]”, que é o terceiro parâmetro passado

GCC

- Consider the code below: /Code/03_GCC/Ex1/main.c:

```
~/Code/03_GCC/Ex1/ $ gcc main.c -o ola_args  
~/Code/03_GCC/Ex1/ $ ./ola_args  
argc = 1  
arg[0]: ./ola_args
```

```
#include
```

```
int main(int argc, char **argv)  
{  
    int i;  
    printf("argc = %d\n",argc);  
    printf("arg[0]: %s\n", argv[0]);  
    if(argc>1)  
        printf("arg[1]: %s\n", argv[1]);  
    if(argc>2)  
        printf("arg[2]: %s\n", argv[2]);  
    return 0;  
}
```

GCC

- Consider the code below: `/Code/03_GCC/Ex1/main.c:`

```
~/Code/03_GCC/Ex1/ $ ./ola_args ABCDE  
argc = 2  
arg[0]: ./ola_args  
arg[1]: ABCDE
```

```
#include
```

```
int main(int argc, char **argv)  
{  
    int i;  
    printf("argc = %d\n",argc);  
    printf("arg[0]: %s\n", argv[0]);  
    if(argc>1)  
        printf("arg[1]: %s\n", argv[1]);  
    if(argc>2)  
        printf("arg[2]: %s\n", argv[2]);  
    return 0;  
}
```

GCC

- Consider the code below: `/Code/03_GCC/Ex1/main.c:`

```
#include
```

```
int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

```
~/Code/03_GCC/Ex1/ $ ./ola_args 12345
argc = 2
arg[0]: ./ola_args
arg[1]: 12345
```

GCC

- Consider `~/Code/03_GCC/Ex1/ $./ola_args 12345 ABCDE` in.c:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;
    printf("argc = %d\n",argc);
    printf("arg[0]: %s\n", argv[0]);
    if(argc>1)
        printf("arg[1]: %s\n", argv[1]);
    if(argc>2)
        printf("arg[2]: %s\n", argv[2]);
    return 0;
}
```

GCC

- Considere o código abaixo, ~/Code/03_GCC/Ex1/main.c:

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

```
int main(int argc, char **argv)
```

```
{
```

```
    int i;
```

```
    printf("argc = %d\n", argc);
```

```
    printf("arg[0]: %s\n", argv[0]);
```

```
    if(argc>1)
```

```
        printf("arg[1]: %s\n", argv[1]);
```

```
    if(argc>2)
```

```
        printf("arg[2]: %s\n", argv[2]);
```

```
    return 0;
```

```
}
```

Ou seja, através dos argumentos “argc” e “argv” recebemos os parâmetros de entrada digitados pelo usuário no terminal

GCC + make

- Considere os códigos abaixo:

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"

int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

```
#include <stdio.h>

void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

GCC + make

- Considere os códigos abaixo:

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"
```

```
int main(int argc, char *argv[])
{
    if(argc<2) return 1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

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

```
void ola(char *hospede);
```

```
void tchau(char *hospede);
```

```
void mudacao(char *hospede);
```

```
#include "mensagens.h"
```

Veremos agora como criar projetos mais complexos, **com vários arquivos** (modular), e como organizar tudo com um Makefile

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"
```

```
int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

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

```
void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

Repare que a função “main()”
chama duas funções, “ola()” e
“tchau()”, que não foram
definidas aqui

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"
```

```
int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

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

```
void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

Antes de começar a função
“main()”, o arquivo
“mensagens.h” foi incluído

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"
```

```
int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

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

```
void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

“#include” significa que todo o conteúdo do arquivo indicado aqui deve ser lido e compilado pelo GCC antes de continuar a compilação deste código

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"
```

```
int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

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

```
void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

Quando se usa as aspas duplas, estamos dizendo para o compilador que o arquivo a ser incluído está na mesma pasta do código atual

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"

int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

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

```
void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

Quando se usa os símbolos “<” e “>”, estamos dizendo para o compilador que o arquivo a ser incluído está em uma das pastas que ele normalmente procura, o que depende da sua configuração

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"
```

```
int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

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

```
void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

O arquivo "mensagens.h" foi incluído, mas ele não diz como "ola()" e "tchau()" funcionam. Ele simplesmente declara que essas funções existem, e quais são seus parâmetros de entrada e de saída

GCC + make

```
#include "mensagens.h"

void ola(char *hospede)
{
    msg("Hello", hospede);
}

void tchau(char *hospede)
{
    msg("Adios", hospede);
}

void msg(char *saudacao, char *hospede)
{
    printf("%s, %s!\n",
           saudacao,
           hospede);
}
```

mensagens.c

A **implementação** das funções “ola()” e “tchau()” foi feita no arquivo “mensagens.c”.

Ambas chamam a função “msg()”, mudando somente o parâmetro “saudacao” desta função

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"

int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

```
#include <stdio.h>

void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

```
#include "mensagens.h"

void ola(char *hospede)
{
    msg("Hello", hospede);
}
void tchau(char *hospede)
{
    msg("Adios", hospede);
}
void msg(char *saudacao, char *hospede)
{
    printf("%s, %s!\n",
           saudacao,
           hospede);
}
```

mensagens.c

GCC + make

```
#include <stdio.h>
#include <stdlib.h>
#include "mensagens.h"
```

```
int main(int argc, char **argv)
{
    if(argc<2) return -1;
    ola(argv[1]);
    tchau(argv[1]);
    return 0;
}
```

main.c

```
#include "mensagens.h"
```

```
void ola(char *hospede)
{
    msg("Hello", hospede);
}
```

```
void tchau(char *hospede)
{
    msg("Tchau", hospede);
}
```

```
void msg(char *saudacao, char *hospede)
{
    printf("%s, %s!\n",
        saudacao,
        hospede);
}
```

mensagens.c

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

```
void ola(char *hospede);
void tchau(char *hospede);
void msg(char *saudacao, char *hospede);
```

mensagens.h

COMO JUNTAR TUDO ISSO?


GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```

Makefile

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```




Ao final da compilação, teremos um executável chamado “mensagens”. Ele depende de dois *objetos*: “main.o” e “mensagens.o”

Makefile

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```



Makefile

A criação do *objeto* “main.o” depende dos arquivos “main.c” e “mensagens.h”

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```

Makefile

A opção “-c”
manda o “gcc”
criar o *objeto*
“main.o”

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```

Makefile

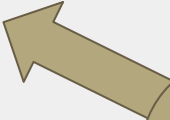
A criação do *objeto* “mensagens.o” depende dos arquivos “mensagens.c” e “mensagens.h”

A opção “-c” manda o “gcc” criar o *objeto* “mensagens.o”

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```

Makefile



Depois de criados
“main.o” e
“mensagens.o”, o “gcc”
“junta” os dois em um
executável chamado
“mensagens”

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
```

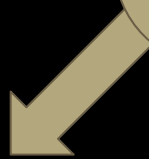
```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ make
gcc -c main.c
gcc -c mensagens.c
gcc -o mensagens main.o mensagens.o
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ ./mensagens Fulano
Hello, Fulano!
Adios, Fulano!
```


GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
```

```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h
~/Code/03_GCC/Ex2/ $ make
gcc -c main.c
gcc -c mensagens.c
gcc -o mensagens main.o mensagens.o
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ ./mensagens Fulano
Hello, Fulano!
Adios, Fulano!
```

O “make” compilou
“main.c” e
“mensagens.c”, e
depois juntou os dois
no executável
“mensagens”. Repare
que também foram
criados os objetos
“main.o” e
“mensagens.o”

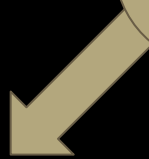


GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
```

```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h
~/Code/03_GCC/Ex2/ $ make
gcc -c main.c
gcc -c mensagens.c
gcc -o mensagens main.o mensagens.o
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ ./mensagens Fulano
Hello, Fulano!
Adios, Fulano!
```

Depois que os objetos foram criados, eles também podem ser reaproveitados em outros códigos



GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
```

Se você fornecer os arquivos **“mensagens.c”** e **“mensagens.h”**, estará trabalhando com código **aberto** (biblioteca de código aberto)


```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h
~/Code/03_GCC/Ex2/ $ make
gcc -c main.c
gcc -c mensagens.c
gcc -o mensagens main.o mensagens.o
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ ./mensagens Fulano
Hello, Fulano!
Adios, Fulano!
```

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
```

```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h
~/Code/03_GCC/Ex2/ $ make
gcc -c main.c
gcc -c mensagens.c
gcc -o mensagens main.o mensagens.o
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ ./mensagens Fulano
Hello, Fulano!
Adios, Fulano!
```

Se você fornecer os arquivos **“mensagens.o”** e **“mensagens.h”**, estará trabalhando com código **fechado** (biblioteca de código fechado)




GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
```

```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h
~/Code/03_GCC/Ex2/ $ make
gcc -c main.c
gcc -c mensagens.c
gcc -o mensagens main.o mensagens.o
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ ./mensagens Fulano
Hello, Fulano!
Adios, Fulano!
```

Reaproveitar os arquivos “**mensagens.c**” e “**mensagens.h**” em outros códigos é muito mais fácil do que recortar e colar as funções de seu interesse em outro código



GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```


Makefile

```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ make clean
rm -f *.o mensagens
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h  README.md
```

GCC + make

```
mensagens: main.o mensagens.o
    gcc $(CFLAGS) -o mensagens main.o mensagens.o
main.o: main.c mensagens.h
    gcc $(CFLAGS) -c main.c
mensagens.o: mensagens.c mensagens.h
    gcc $(CFLAGS) -c mensagens.c
clean:
    rm -f *.o mensagens
```

Makefile



```
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.o
main.o  mensagens  mensagens.h  README.md
~/Code/03_GCC/Ex2/ $ make clean
rm -f *.o mensagens
~/Code/03_GCC/Ex2/ $ ls
main.c  Makefile  mensagens.c  mensagens.h  README.md
```